A Generic User Interface for Energy Management in Smart Homes

Zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der KIT-Fakultät für Wirtschaftswissenschaften
Karlsruher Institut für Technologie (KIT)

genehmigte
Dissertation
von

M.Eng. (Comp. Sc.) Huiwen Xu

Tag der mündlichen Prüfung:	18. 12. 2018
Referent:	Prof. Dr. Hartmut Schmeck
Korreferent:	Prof. Dr. Andreas Oberweis

Abstract

Considering the current energy crisis and the serious need for environmental protection, reducing energy consumption or, expressed differently, improving energy efficiency, has become a global concern, with particular attention being paid to the saving of fossil fuels and the cutting down on environmental disruption. In order to deal with this challenge, electricity generated by renewable energy sources, such as wind and solar energy, is being integrated into power grids, thereby reducing the dependence on conventional power plants. Since renewable energy sources are characterized by higher degrees of uncertainty, are subject to fluctuation and are by nature decentralized, power grids are facing a major transformation. This transformation is driven not only by the need to integrate the increasing share of renewable energy but also by the need to improve energy efficiency and allow consumers more control over their energy consumption. One contribution which could solve some of the problems arising from renewable energy sources in the power grid, could be the energy management in household buildings, since it would enable load shifting.

By equipping a household building with a building operating system, a smart meter and intelligent appliances, the building can be turned into a smart home. Building operating systems can manage heterogeneous household devices by means of Internet of Things (IoT) technologies and realize the in-house energy management by using their own solution or providing a universal runtime environment, in which energy management applications can be deployed. Therefore, building operating systems play an essential role in shifting the electricity demand and improving the energy efficiency

in household buildings. With an increasing need for on-demand flexibility in the power grid, it is plausible to assume that, in the near future, an increasing number of providers will be attracted toward offering building operating systems. However, this increasing number of competing building operating systems will make a holistic, comprehensive overview of the energy flows and devices in household buildings more difficult. In a fragmented market, customers might lose control over their individual targets concerning their building. From this arises the need for a flexible and full-featured user interface which can visualize the energy data of a building and allow residents to be able to collect and communicate individual needs and preferences to the building operating system.

This thesis proposes a generic user interface for building operating systems, which, on the one hand, is able to communicate with building operating systems in diverse smart homes and, on the other hand, can provide residents with holistic and transparent information about the energy consumption and generation in their building as well as offer multiple types of control over their appliances. Current building operating systems are basically equipped with one or more proprietary user interface(s) for their application scenarios based on their specific Application Programming Interfaces (APIs). An architecture for a generic user interface, which enables it to be compatible with these heterogeneous systems, has been proposed in this thesis. The key part of the architecture, which makes the user interface generic and extendible, is composed of a number of abstract data models. These data models define the logical structure of the functional units needed by the generic user interface, as well as the relationship to each other. To ensure that the user interface can be flexibly adapted to diverse types of buildings, the data models have been designed to be independent of any building operating system. In addition to this, three roles, namely, the administrator, the operator, and the resident were introduced in the thesis in order to facilitate security administration and division of responsibilities in household

buildings. Furthermore, a variety of use cases related to smart homes were collected and presented. As a result, a number of functional components were proposed to be supported by the generic user interface so as to enrich the features of the user interface to be able to comply with more of these use cases.

Based on the design outlined in the thesis, a prototype of a generic user interface named 'Building Operating System User Interface (BOS UI)' was developed and implemented as the user interface for the building operation system, the Organic Smart Home (OSH), which is installed in the Energy Smart Home Lab (ESHL) at the Karlsruhe Institute of Technology (KIT). The detailed functions that were implemented in the BOS UI have been described in the thesis. Subsequently, the BOS UI was evaluated qualitatively and quantitatively by combing theoretical analysis and experiments. The evaluation results show that the BOS UI basically meets a set of desired requirements for a generic user interface of building operating systems, and the evaluation experiments yielded very positive feedback in many aspects including improvement of energy efficiency and user experience. The thesis concludes by proposing corresponding solutions for improvements of the current design in the future.

Therefore, the work of this thesis contributes to the field of energy informatics by firstly proposing an architecture, roles, data models and functional components for a generic user interface for building operating systems, secondly, by implementing a prototype of this generic user interface, and finally, by evaluating the design, functionality and usability of the generic user interface by combining both a qualitative and a quantitative analysis.

Acknowledgement

The four years during which I have been engaged in my doctoral project in Germany, have become an important turning point in my life and the memory of this journey is so valuable and unforgettable that it will stay with me for the rest of my life.

Foremost, I would like to express my sincere gratitude to my supervisor Prof. Dr. Hartmut Schmeck who trusted me and gave me the opportunity of pursuing my Ph.D. in his research group. Firstly, I appreciate that Prof. Schmeck provided me with a favorable platform and a flexible research environment for my Ph.D. project. Secondly, I am grateful that Prof. Schmeck afforded me his huge on-going support whether emotionally or financially throughout the long period during which I was working on the project. Without his guidance and feedback, I could never have completed this thesis. Finally, I thank Prof. Schmeck for having such a positive impact on my work. I had the privilege of closely working together with Prof. Schmeck, during my time as his teaching assistant. In addition to his profound knowledge, I was impressed greatly by his accuracy in teaching, his sense of responsibility toward students, and his open mind to embrace external criticisms. All of these have had a great influence on me.

In addition, I would also like to thank my colleagues at KIT and FZI for discussing with me, inspiring me with new ideas and giving me feedback concerning this work. A special thank you to Kaibin, who sacrificed so much personal time to have brainstorms with me and gave me so many constructive suggestions throughout my Ph.D. study; to Christian, who helped

me get my work on the right track during the initial phase of the project by offering me so many insightful comments; to Jan and Ingo, who assisted me in deploying the UI prototype for this work to the ESHL and set up a simulation environment for the purpose of the evaluation experiments; and last but not least, to Lukas and Doris, who lent me a hand with designing and organizing the evaluation experiments and helped me to finalize the thesis. Many thanks also go to Miki, Friederike, Fabian, Marlon, Birger, Florian, Sebastian, Fredy, Kevin, Mischa, Pradyumn, and many more colleagues who have given me a lot of help during my Ph.D. study.

Furthermore, I would like to thank my friends who kept me company and lent me emotional support in the process of realizing my dream in Germany. I am especially grateful to Sai, a great friend, whom I have known for more than ten years, for always being around me, talking to me, and sharing feelings with me. To Kenan, who gave me the most inspiration for living a happy life, for helping me find the inner peace to deal with pressure and depression in life. To Stephan, Yingzhao, Rudi, Chunyan, Yaoyao, Ningyi, Siqi, Xinshuang, Chong and many more friends that I met in Germany for making my life better than it was. My special gratitude is given to those wonderful friends in the free evangelical community (FeG) of Karlsruhe. I did not realize how incredibly nice people could be until I joined this community. They have given me the feeling of home in a foreign country. Particularly, I would like to express my sincere gratitude to Sabine, who is not only so nice as to offer me help in proofreading and correcting this thesis, but more importantly, she has made me really feel the power and the charm of one who has a firm faith.

Finally, with all my heart, I would like to thank my parents and my sister for always being my rock, and for their continuous encouragement, eternal care and love. I could not have accomplished this work without them.

Karlsruhe, May 2018 *Huiwen Xu*

Contents

1 Introduction

In the face of a global energy and environmental crisis, the global power grid is continuously being challenged to gradually undergo a transition from fossil-based energy sources to renewable energy sources. This transition poses a threat to the stability of the power grid. Smart homes and buildings which have potential of providing demand flexibilities can contribute toward balancing the demand and supply in the grid. The user interface of a smart home plays an essential role in realizing demand response, since it helps to increase the residents' energy-awareness and facilitates load shifting by allowing residents to configure degrees of freedom for their appliances. After outlining a motivation, this chapter continues by laying down some basic assumptions which were made for this thesis and could apply to the smart home of the future. After this, some research questions are stated, which will be addressed later in the thesis. It concludes by highlighting the major contributions of the work and then outlining the content structure of the thesis.

1.1 Motivation

According to the latest World Population Projections[1], 7.6 billion people inhabit the earth at present, and in the next few years, the number is estimated to keep increasing by around 80 million per year. The growth of the global population creates a rising demand for energy supply which on

[1] http://www.worldometers.info/world-population/world-population-projections/

the one hand leads to the excessive exploitation of natural resources and on the other hand results in increasingly serious environmental pollution, and enhancing the climate crisis. During the 20 years between 1984 and 2004, primary energy consumption grew by 49% and CO_2 emissions by 43%, with an average annual increase of 2% and 1.8%, respectively [94]. It is known from IEA Statistics[2] that 80.8% of the total global energy consumption in 2015 was derived from fossil fuels, which are non-renewable resources and will generate a large amount of greenhouse gases after combustion. Because of this, the reduction of energy consumption has become a global concern, with particular attention being paid to the saving of non-renewable natural resources and the cutting down on environmental disruption.

To cope with the challenge, the European Union (EU) in 2007 set up "20-20-20" climate and energy targets for its Member States for the year 2020, which are 20 percent reduction of EU greenhouse gas emissions, 20 percent share of renewables of overall EU energy consumption, and 20 percent increase in energy efficiency compared to 1990. However, official data for 2015 shows that the overall target level of Member States is still insufficient: the sum of the 28 national targets for 2020 primary energy consumption does not match the reduction target determined at the EU level [8]. It is clear that energy saving is still a big challenge which needs to be addressed. In 2016, the households which use energy for the purposes of space and water heating, space cooling, cooking, lighting and electrical appliances and other end-uses represented 25.4% of the total energy consumption in the EU [6]. This indicates that improving energy efficiency in household buildings will have a positive effect on reducing the overall energy consumption and the carbon footprint.

Smart homes, being equipped with smart appliances and smart meters, can be used as a solution to improve energy efficiency of household buildings by displaying various energy parameters (e.g. energy price, devices'

[2] https://data.worldbank.org/indicator/EG.USE.COMM.FO.ZS

power consumption information, etc.) for residents and allowing them to control their energy usage in intelligent ways. Compared to conventional buildings, the feature of energy visualization provided by the user interface of smart homes is able to improve the residents' awareness and understanding of the energy consumption in their building, which will then influence them to change their behavior towards energy savings. It was found that, displaying real-time information on electricity usage to users, leads them to effectively modify their behavior to achieve an energy saving of up to 30% [78, 58].

Another effective way to conserve natural resources is to integrate renewable energy into the power grid so as to reduce the dependence on the power generated by conventional, fossil fuel-fired plants. In contrast to fossil fuels, renewable energy sources, such as solar energy and wind energy, are not only constantly being replenished but can also be made use of without producing pollution or contributing to global warming. However, since renewables are usually fluctuating, decentralized, and uncontrollable, their integration into the power grid poses a threat to its stability, therefore there is an increasing need for on-demand flexibility in the power grid in order to maintain the balance between supply and demand at short notice.

One contribution to ease the imbalance brought by the increasing share of renewable energy sources in the power grid, is energy management in household buildings, which is one of the features of a smart home. A wide variety of information from different entities is needed in order to realize demand side response in a smart home. The external signals (e.g. time-of-use energy tariffs) from utilities can provide an indication of peak times or off-peak times in the power grid. The building operating system, which is a software deployed on smart homes, can either be equipped with its own energy optimization algorithm or provide a platform for other energy management applications to analyse current and predicted external signals and optimize the in-home energy use, by shifting the power consumption in the

building from peak periods to off-peak periods. The load shifting can be realized by shifting the load profiles of re-schedulable appliances (e.g. dish washers or washing machines). Since it is important to respect the residents' needs, in order to ensure their comfort, each resident needs to specify degrees of freedoms for their appliances via the user interface of the building operating system in their building.

The user interface is an indispensable component of the building operating system in a smart home, since it is essential in order to keep the residents' personal preferences in the loop. Without this human interaction, the building operating system cannot work properly. The user interface is not only able to display the external signals, such as time-variable prices for residents, but also provides transparent information about the energy consumption as well as the energy generation of a particular household. This, in turn, enables the residents to develop an awareness of whether they are using electricity in a reasonable way or not and so help them to develop good habits for saving energy and money. In addition to this, the user interface makes it possible to improve the comfort and convenience for residents of a smart home by allowing them to give input to the automatic control of the appliances in their building. As mentioned earlier, although building operating systems can help to optimize schedules of household appliances so as to achieve the purpose of the demand response, they still need to respect residents' needs and thus ultimately be under the control of the residents. For this reason, the user interface of the building operating systems not only provides the residents with time options for setting degrees of freedom for their appliances but also offers control options which allow residents to alter the appliance scheduling any time they want.

Currently, smart appliances are becoming more and more popular on the market. Many big brands have launched their own smart home appliances and platforms in order to build their own smart home ecosystems. However, the acquisition of smart home technology has been relatively slow since the

development of smart homes is still in its infancy. One of the largest obstacles that hinder the market popularization of smart homes is the lack of a unified standard of communication. Vendors of smart appliances all try to build their own platforms as they populate the market. This has led to a variety of incompatible standards on the market. Under these circumstances, in order to make their homes smart, consumers either have to buy all the household appliances for their building from the same brand, which is expensive and unrealistic for most of consumers and few of them can accept it, or consumers need to install various applications together with their respective user interfaces in order to realize the intelligent control of their appliances. In the latter case, consumers then have to bother about dealing with multiple inconsistent user interfaces from different vendors.

Some building operating systems have already been implemented, which try to work around this dilemma by introducing a unified hardware abstraction layer that shields the discrepancies between the various protocols and devices. The hardware abstraction layer is able to convert the protocol related data to the unified data models used by the building operating system, and depending on the particular data models, each of the building operating systems is equipped with one or more proprietary user interface for the interaction with users. However, these user interfaces currently still have a lot of room for improvement. Since they are tightly coupled with their corresponding building operating system, the functionalities they can provide are also restricted by their underlying building operating system. This greatly limits the number of use cases in smart homes. However, since the current building operating systems are still in their early development phase, their time for going on the market is a long way away. Much research will still need to be done in order to overcome various difficulties and challenges of the systems.

Current publications (e.g. [91, 52, 77, 96, 80]) reveal that most of the research about user interfaces for building operating systems done in the past

years has dealt primarily with method description, which is more theoretical in nature. So far, the publications in the area of practical implementation of the user interface for smart buildings are very few. Because of this, it is desirable and valuable to not only design but also implement a user interface, which on the one hand is generic and flexible enough to be able to interact with users as well as communicate with heterogeneous building operating systems, and, on the other hand, is able to support a wide range of use cases relating to smart homes while at the same time ensuring good usability.

1.2 Assumptions and Research Questions

As previously outlined, the major motivation for the work in this thesis is the aspect of energy saving and on-demand flexibility. With the aid of their user interface, building operating systems can support in-house energy management and provide residents with transparent information on external signals and energy use in their household. This would make a valuable contribution toward improving energy efficiency and realizing demand response. However, as mentioned earlier, currently, building operating systems are still at an early stage of scientific research and have, therefore, not yet been popularized on the market. In order to make the research in this thesis more scientifically rigorous, there are a number of fundamental assumptions that need to be declared in advance.

- It is assumed that, in the future, a great number of conventional power plants (e.g. coal-fired plants and nuclear plants) will be forced to close down. Instead, a large proportion of electricity in the power grid will be coming from renewable energy sources, especially wind and solar energy which are fluctuating, decentralized and hardly controllable sources. As a consequence, the future power grid will face a lot of unprecedented problems such as mismatches between demand and supply, deviations between predicted and actual power generation and

bottlenecks in the low voltage distribution grid. In order to tackle the challenges arising from renewables, high degrees of flexibility in energy demand will be required to ease the imbalance in the power grid.

- Research at the intersection of computer science and economics has revealed that incentives are a powerful way to allocate scarce resources [102, 57]. Therefore, it is assumed that the rising need for demand flexibility in tomorrow's power grid will bring with it new incentives to promote load shifting. The incentives could be given in the form of price signals, i.e., time-variable electricity prices, or incentive payments in order to shift the load from peak times to off-peak times.
- It is assumed that the conventional meters used in household buildings today, will, in the future, be replaced with smart meters which enable two-way communication between the meter and a utility. In other words, the smart meters are able to not only measure and record energy consumption during a certain time interval but also receive tariff information from the energy utility. In addition to smart meters, household buildings in the future will also be equipped with either smart appliances which can be remotely monitored and controlled or sensors and actuators that indirectly make normal appliances "smart" by enabling remote interaction with them.
- The high incentives for load shifting and the public's willingness to make their homes smart will, in the future, attract a lot of providers of various building operating systems, including energy management systems, which will make full use of the potential of smart meters and smart appliances in a building, in order to achieve certain goals. In facing the numerous incompatible communication standards on the market, these building operating systems will be capable of providing their unified platform for higher level applications.

- It is assumed that building operating systems in the future will focus on various energy carriers (e.g. electricity, heat, gas, etc.) and support different functionalities. In order to be able to interact with users, the building operating systems will be equipped with their own, customized user interface. The rising number of building operating systems, some of which may be competing with each other will make a holistic and comprehensive overview of the energy flows and devices in a household building more difficult.
- Due to the lack of unified standards, it is assumed that the smart home market will be in a state of technological fragmentation for a long time in the future. In such a fragmented market, customers might easily lose control over their individual targets concerning their building.

In addition to the aforementioned assumptions, a delimitation will also need to be made beforehand in order to define the research scope of the thesis. The user interfaces used for this thesis are all designed for building operating systems which are deployed in household buildings. A household building, in this thesis, refers to either one of the apartments in a larger residential building or a free standing single or multi-storied house which is owned by one person or family. An important feature for the "building" in this thesis is that there is one person who is responsible for paying all the energy costs. Residential buildings that are shared by a number of different home owners, office buildings, commercial buildings and other public buildings are beyond the scope of this thesis, but the concepts presented in this thesis are easily extended to these scenarios.

According to the current development of smart home technologies, one of the challenges that the smart home faces is that it cannot provide users with a convenient and unified user interface at the application level. This will greatly affect the development of the smart home market. In order to

lift the smart home out of this dilemma, the core research question that this thesis is going to answer is as follows:

> **Research Question:** *How should a generic user interface be designed, so that it can deal with different kinds of building operating systems while ensuring good usability?*

The core objective of this thesis is to design and implement a generic user interface for building operating systems, which is not coupled with any of the existing building operating systems. The main tasks of the generic user interface are:

i to increase the comfort of household buildings by allowing residents to realize all kinds of home automation,

ii to provide transparent information on energy consumption and generation in the household, so as to raise the residents' awareness about the energy use in their building and generate financial savings,

iii to identify the degrees of freedom of the appliances in the building for the building operating systems so that the load profiles of the appliances can be shifted from peak to off-peak periods.

A valuable user interface is one that is not only able to provide rich functionality but is also highly user friendly since the latter plays a crucial role in determining the user's experience of the user interface. Therefore, the final implementation of the generic user interface needs to be evaluated by test objects and its usability is expected to be good. The core research question can be further divided into the following sub-questions, as considered from different perspectives.

> **Sub-question 1:** *What is the definition of a generic user interface for building operating systems?*

Since a generic user interface for building operating systems is the work that has been done for this thesis, it is of great importance to define what this is very clearly before starting on the design. The user interface as a general term has been well defined in many publications. However, specifically, when it comes to a generic user interface for building operating systems, there has not been any published definition available for the concept. In this case, the thesis defines the term by specifying and analysing a series of conditions that need to be met.

> **Sub-question 2:** *What does the architecture have to look like for such a generic user interface?*

The architecture which defines the system elements and how they interact is an abstraction of a system and reflects earliest design decisions about a particular system [90]. A generic user interface is capable of exchanging information not only with different building operating systems but also with some external entities. To this end, its architecture needs to incorporate appropriate components which can bridge the gap between the generic user interface and its underlying building operating systems and allow communication among different stakeholders.

> **Sub-question 3:** *What are data models for such a generic user interface?*

Data models that create structures of the data used by the generic user interface and standardize their relationship to each other play an essential role in making the user interface independent of a specific building operating system and translating user targets to building operating systems. The data models in the back-end of the generic user interface should be abstracted

properly so that the user interface can be generic and sufficiently flexible to deal with different building operating systems while not losing effectiveness.

> **Sub-question 4:** *What are the functional components that the generic user interface should cover?*

The aforementioned tasks of the generic user interface for building operating systems can be further subdivided into a number of sub-tasks which need to be supported by different functionalities. In order to complete the tasks to the maximum extent, this thesis provides answers to this research question by proposing a set of functional components to support a variety of use cases. The functionalities supported by these components have been implemented as well and they are also evaluated by a group of test objects at the end.

> **Sub-question 5:** *How can such a generic user interface be made configurable and extendible?*

In order to facilitate user-friendliness, it is favorable to a user interface if it is able to be configured or customized by users according to their preferences. Different aspects of the generic user interface need to be configurable. For example, one important aspect might be allowing users to set languages, which is beneficial for users who prefer to use their native language. In addition, extendibility is also a valuable feature for the generic user interface. Since household buildings are basically diverse from each other and the appliances used are also multifarious, it is important that the generic user interface is extendible so that it can be applied to various buildings and adapted to different situations.

1.3 Major Contributions

The major contribution of this thesis is the design, implementation and evaluation of a generic user interface for building operating systems, which mainly focuses on two dimensions relating to the field of smart homes, namely, comfort and energy.

To date, there are a few building operating systems which are in the developmental stage. However, since the goals and functionalities provided by these building operating systems differ from one another, their user interfaces are also very different. A first step in designing a better user interface would be a careful review and evaluation of state-of-the-art user interfaces used in current building operating systems. Since this type of comparative review and evaluation could not be found in existing publications, this will serve to provide the first contribution of this thesis. To begin with, then, the thesis introduces a number of open-sourced building operating systems and analyses the working mechanism of their user interfaces. Subsequently, a series of use cases relating to a smart home is proposed on the basis of research that has been done as well as scenarios of the existing building operating systems. Furthermore, the thesis evaluates the user interfaces that have been reviewed according to two aspects. One aspect is from the use cases, and another aspect is based on the technical characteristics of the user interfaces. To this end, a list of technical evaluation criteria is proposed. The final evaluation results indicate both the strengths and weaknesses of these user interfaces. None of them can cover all of the evaluation criteria. They all leave room for improvements of varying degrees in different aspects. [107]

The current building operating systems are basically equipped with their own proprietary user interface for their application scenarios, based on their specific application programming interfaces (APIs). In the design phase of this thesis, an architecture for a generic user interface for building operating

systems has been proposed. The central part that, enables the generic user interface to uncouple itself from its underlying building operating system is a set of generic data models, which are appropriate abstractions of objects that are needed by the generic user interface. The existing building operating systems with their own proprietary data models need an adapter to convert their data models into the generic data models, so that the generic user interface can work with them. The building operating systems of the future will be able to directly use the generic user interface if the generic data models are adopted into their system. The data models proposed in this thesis are generic and flexible enough to cover hybrid energy sources in households (e.g. electricity, heat, cold and gas), serve various buildings and support different advanced home automation services. In order to facilitate security administration and division of responsibilities, this thesis introduces three roles, namely, the administrator, the operator and the resident, for the context of a smart home. Each role is given different permissions to accomplish different tasks. At the end of the design phase, a number of functional components supported by the generic user interface are listed and described. This provides answers to the research sub-question 2.

Based on the aforementioned design, a prototype of the generic user interface, named "Building Operating System User Interface (BOS UI)" is implemented and applied to a building operating system, the Organic Smart Home (OSH), which has been installed in the Energy Smart Home Lab (ESHL) at the Karlsruhe Institute of Technology (KIT). The ESHL is an apartment with two bedrooms that is not only equipped with basic, measurable and controllable household appliances (e.g. light and washing machine) but also has an electricity generating system (e.g. PV panels) as well as a cogeneration system (e.g. micro Combined Heat and Power Plant (μCHP)). Since the design of the BOS UI is independent of the OSH, a couple of extra components are introduced in order to achieve the two-way communication between the BOS UI and the OSH. The BOS UI is developed on the basis of role-based

access control. After the connection with the OSH, the BOS UI can accept a series of configurations in order to adapt the user interface to the environment of the ESHL. For instance, the administrator can define the layout of the ESHL on the BOS UI and allocate devices to appropriate locations, and the operator can manage residents by assigning them permissions to access the devices in the ESHL. After configuration, the BOS UI can provide residents with a holistic overview of the energy flow in the ESHL as well as provide intuitive options for specifying degrees of freedom of appliances in order to help improve energy efficiency. In addition to that, the BOS UI also supports all kinds of home automation services to maximize the comfort of residents. Furthermore, the BOS UI has a responsive layout and is configurable in many aspects, including themes, layouts, languages, etc. which all contribute toward good usability.

The last contributions of the thesis are the final evaluation experiments and the results derived from them. Besides the BOS UI, the ESHL GUI, which is the original user interface of OSH in ESHL, has also been used for the evaluation in order to have a comparison. To evaluate the functionalities as well as the usability of the two user interfaces, experiments have been conducted by inviting a group of test users to execute a number of predetermined tasks, using the user interfaces, and then asking the test users to fill out questionnaires at the end. This thesis compares the functionalities of the BOS UI with those of the original ESHL GUI, from various perspectives, based on the statistical results of the experiments. To determine usability, a robust and reliable evaluation tool, named System Usability Scale (SUS), has been used in the experiments in order to get a quick and clear statement about the usability of the two user interfaces. According to the test users' feedback, as received in the questionnaires, the BOS UI has made great improvements with regard to both functionality and usability in comparison to the ESHL GUI, and besides it is also able to facilitate the improvement of energy efficiency in household buildings. What is more, many test users

made valuable comments, some giving helpful criticism and/or advice concerning the two user interfaces, which may help to detect defects in the current systems but are also worth considering when designing user interfaces for building operating systems in the future.

1.4 Thesis Outline

The following chapter, Chapter 2, introduces some background information needed for this thesis. It firstly describes and analyses the functionalities of a smart home and then gives a brief introduction to a few open-sourced building operating systems. After that, it reviews the development of graphical user interfaces and outlines a number of basic principles for user interface design. Chapter 3 gives an overview of state-of-the-art user interfaces for building operating systems and also evaluates each of these user interfaces according to two aspects, namely, their use cases and their technical characteristics. To start the design of a generic user interface for building operating systems, a definition of the concept is given in Chapter 4, and subsequently the chapter proposes an architecture for a generic user interface that can deal with different building operating systems. Subsequently, three "roles" and their corresponding "permissions" are defined, in order to ensure the security and privacy protection of the inhabitants. The chapter further describes a set of data models and functional components used in the generic user interface. Chapter 5 focuses on the implementation of the generic user interface. Since the prototype of the generic user interface was to be applied in the Energy Smart Home Lab (ESHL), a brief description of the ESHL is given at the beginning of the chapter. A few more components are introduced after that, which allow the prototype of the generic user interface to be connected to the building operating system of the ESHL. Finally, the chapter proceeds to illustrate the detailed functions that have been implemented in the prototype. Chapter 6 evaluates the prototype of the user interface that

is implemented in Chapter 5 according to three aspects - design, functionality and usability. The evaluation of the design of the user interface is based on theoretical analysis. Functionality and usability are evaluated by inviting test users to participate in experiments. Chapter 7 concludes the thesis with a summary of the findings made, and gives an outlook for future work.

1.5 Previous Publications

This thesis summarizes the results of several years of research. During this time, several papers were published and presented as part of journal and conference contributions. Most of the chapters of this thesis are based upon these previously published papers.

The introduction to the Energy Flexibility Platform and Interface (EF-Pi) in Chapter 2 is based on the joint work with Christian Gitte, Fabian Rigoll, Joeri van Eekelen and Michael Kaisers. This part has been presented in the 5th D-A-CH+ Energy Informatics Conference in conjunction with 7th Symposium on Communications for Energy Systems [67]. Chapter 3 reviews the state-of-the-art user interfaces for building operating systems and evaluates them with respect to different dimensions, which has been partially presented in the 2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC 2017) [107]. At the beginning of Chapter 5, an introduction to the Energy Smart Hone Lab (ESHL) is provided. As part of the introduction, the decentralized service oriented architecture based on a message-oriented middleware in the ESHL has been published in the first International Workshop on Mashups of Things and APIs (MoTA 2016) with Kaibin Bao, Ingo Mauser and Sebastian Kochanneck [44]. The major content of this thesis, namely, Chapter 4, Chapter 5 and Chapter 6 has been published in the Journal of Energy Informatics [106].

2 Fundamentals

This chapter provides the background information relating to the topic of the thesis. Since the smart home is the actual object that the user interface, designed and implemented in the thesis, applies to, this chapter firstly gives a few definitions of a smart home and then introduces the functionalities of smart homes in terms of four aspects - comfort, health, security and energy. The functionalities in a smart home can be achieved with the aid of building operating systems deployed at smart homes. This chapter therefore proceeds to outline the concept of a building operating system and gives a few examples thereof. After providing the background information related to the application field of the user interface presented in the thesis, a brief review of the history of graphical user interfaces is given in order to highlight the trends and patterns of user interfaces over the years. The chapter concludes by listing and outlining a number of principles used in user interface design. These principles will also be used for guiding the design and implementation of the user interface in the thesis.

2.1 Smart Home

In the book, 'The Road Ahead', which was published in 1995, Bill Gates describes his futuristic house that was being built on Lake Washington outside of Seattle in Medina. He had hoped that his private home of the future would consist of silicon and software and would be able to assimilate constantly updated cutting-edge technologies. The house was finally completed in 1997. It was equipped with fibre optic cables and every door, window,

lamp and appliance was able to be remotely controlled with the aid of touch pads. In the back-end, there was a high performance server to manage the whole system. "First thing, as you come in, you'll be presented with an electronic pin to clip to your clothes. This pin will connect you to the electronic services of the house," Gates wrote in The Road Ahead. "The electronic pin you wear will tell the house who and where you are, and the house will use this information to try to meet and even anticipate your needs - all as unobtrusively as possible." Most ordinary people may think that this is too far away from their own lives, however Gates in his book said, "A decade from now, access to the millions of images and all the other entertainment opportunities I've described will be available in many homes and will certainly be more impressive than those I'll have when I move into my house in late 1996. My house will just be getting some of the services a little sooner."

Now twenty years have passed. The development of information and communication technologies in reality gradually confirmed Gates's prophecy. With the rapid growth of Internet of Things (IoT), many countries and corporations have put forward various smart home solutions in order to satisfy peoples' rising demand in the areas of comfort, security, entertainment, energy efficiency, etc. At the Appliance & Electronics World Expo held in Singapore in May 1998, the smart home system of the Singapore model was introduced through the simulation of the "Home of The Future". The functions of the system included remotely connected smart meter, security alarm, video intercom, monitoring center, appliance control, cable access, telephone access, household information message, home intelligent control panel, smart wiring boxes, broadband network access system and software configuration etc. In April 2016, the Versionaire EC[1], which is the first Smart Home Executive Condominium in Singapore, was launched and marketed. The 632-unit executive condominium is specifically designed to embrace the IoT concept. With the aid of a tailored mobile application

[1] http://www.thevisionaires-ec.com.sg/

named HiLife, the Smart Home Control Centre allows residents to control all the smart features in their home at their finger tips, e.g. automated appliances, security monitoring, temperature control, etc. In 2015, Panasonic corporation created a showcase, called Wonder Life-BOX 2020[2], in the Panasonic Center Tokyo. With a total floor space of 400 square meters, the prototype of a private apartment shows what a smart home could look like in the future and therefore presents Panasonic's vision for a quality lifestyle in 2020. A variety of innovative services like lifestyle security support, voice interactive, smart screens integrated with interior design, smart healthcare navigation, etc. are all embraced by the Wonder Life-BOX, which is expected to become a reality soon. In 2014 and 2016, two smart home products, Amazon Echo and Google Home, came out, respectively. These two types of voice-controlled speakers are capable of performing various services, including acting as a home automation hub to control smart devices in a building thereby allowing residents to operate their smart home via voice commands. The two smart speakers are increasingly gaining their popularity since they were first introduced into the market. According to this year's smart audio report from NPR and Edison Research [25], so far around 39 million Americans, which accounts for 16% of the total population of USA, own a smart speaker. The number of users has increased by 128% compared to last year, and 64% of these users buy the speakers to control smart home devices.

Smart homes aim to establish a better quality of living by deploying fully-automated control of appliances and providing assistive services [37, 78]. Since the first smart building in the world was built in the United Sates in 1984, the term of the smart home was first used in an official way by the American Association of House Builders [71]. Henceforward the concept has been widely used in both academia and industry. However, so far, no

[2] https://www.panasonic.com/global/corporate/center/tokyo/floor/lifebox.html

unified definition has been put forward. The following are several definitions that have been collected from recently published literature.

"The Smart Home concept is the integration of different services within a home by using a common communication system. It assures an economic, secure and comfortable operation of the home and includes a high degree of intelligent functionality and flexibility." [79]

"A smart home is a home which is smart enough to assist the inhabitants to live independently and comfortably with the help of technology is termed as smart home. In a smart home, all the mechanical and digital devices are interconnected to form a network, which can communicate with each other and with the user to create an interactive space." [99]

"A smart home can be defined as a residence equipped with computing and information technology which anticipates and responds to the needs of the occupants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and connections to the world beyond." [71]

"A smart home is an application of ubiquitous computing that is able to provide user context-aware automated or assistive services in the form of ambient intelligence, remote home control, or home automation." [37]

In a survey published in 2012 [37], the authors categorized the smart home projects at the time, according to the three intended services: comfort, healthcare and security (cf. Figure 2.1). Broadly looking at the various current definitions of the smart home concept, it would seem that, improving the comfort of the residents, is considered to be the basic goal of the smart home. One of the most important means of achieving this goal, is home automation, which enables lights, blinds, heating, washing machines and other appliances in a building to be automatically controlled via a user interface, thereby saving the trouble of manual operation. The basic home automation can be further enhanced by the introduction of so-called scenes.

Normally more than one device is involved in a scene and each one's target state needs to be specified in advance in the so-called scene definition. A number of devices in a scene can be controlled at the same time when the scene is triggered so that the comfort of residents can be further improved. Another way of extending the basic home automation is by introducing the use of the timed switch. Residents can set a time to the target state of devices in order to realize the timed automation of devices in their building. Activity identification is another feature that is mentioned in the survey [37], which can help ease the residents' daily life. In this case, the smart home needs to be aware of context information in the building in order to be able to make corresponding responses to the changes in the smart home environment. The context information in a building can be classified into four essential categories - identity, location, status (or activity) and time [60]. By collecting and analysing the context data, the context-aware applications or services provided by the smart home are able to automatically adjust the state of proper devices according to different needs so as to maximize the residents' comfort. The following is an exemplified scene, which could happen in a context-aware smart home. When a resident falls asleep while watching television at night, his smart wristband or sleep monitor will detect his sleep state based on the body index and will alert the smart home system, which will then send commands to turn off the television, music player, lights, etc. and shut down the blinds in order to create a good sleeping environment for the resident.

Nowadays, many countries are stepping into an aging society, but, at the same time, the pace of young people's lives is becoming faster and faster. This leads to a growing concern about health and security issues of elderly people, especially of those, who are living by themselves. Because of this, the healthcare service for elderly people, patients or disabled people is another functional dimension of a smart home that has been the focus of many projects recently. Being equipped with all kinds of sensors (e.g. environ-

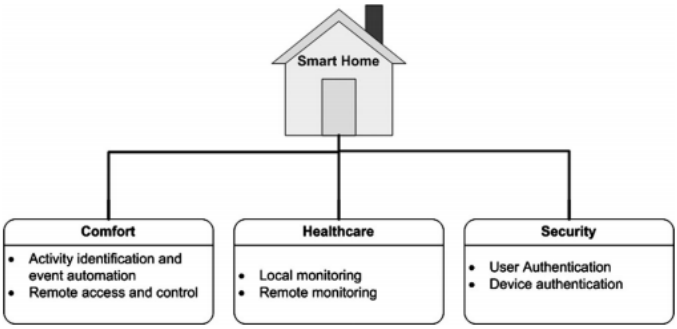


Figure 2.1: Categorization of smart home projects according to the intended services [37]

mental, activity and physiological sensors), smart homes have the potential of providing an ambient assisted living environment for elderly or disabled people, who need to be taken care of. The authors in the above mentioned survey [37] classified the healthcare services provided by smart homes into local monitoring and remote monitoring services. Through the service of local monitoring, the patients' various vital signs, including physiological parameters like heart rate, body temperature, respiration rate and blood pressure, etc. are monitored and recorded in smart homes, and can gradually reflect a health trend of the patients which can later be analysed by health care providers. In addition to this, any activity disorders such as falls, immobility, reaction incapacity, etc. can equally be detected in a timely manner and local warnings or alarms can be generated in smart homes to inform the caregivers. The remote monitoring, on the other hand, is mostly used for the rescue of patients in case of an emergency. When patients get involved in an emergency situation, this dangerous situation will be detected by their on-body sensors and an emergency call will be automatically dialled to the nearest emergency service provider. Figure 2.2 shows a smart home solution for elderly healthcare, which was proposed in a recent survey, where

a comprehensive review of the state-of-the-art research and development in smart home based remote healthcare technologies was presented [81].

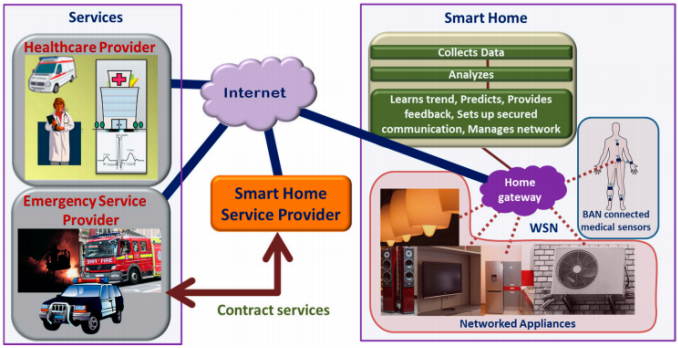


Figure 2.2: Schematic diagram of a smart home showing the network among different stakeholders [81]

Security is one of the most pressing issues of residents these days and, therefore, constitutes another major functional dimension that needs to be taken care of by smart homes. The survey mentioned earlier [37] categorized security in smart homes into user authentication and device authentication, both relating to the software framework of smart homes and having the purpose of eliminating security holes in the framework. To prevent potential security threats from the outside, authentication mechanisms need to be introduced to smart homes in order to secure the access to the smart home network and prevent the unauthorized use of devices. Besides this, firewalls and anti-virus software can be installed in the smart home gateway which is also an effective way of withstanding outside attacks. Considering all kinds of threats in the physical world, home security and safety need also to be ensured in smart homes. To this end, various threat detection and automatic alarm functions (e.g. smoke detection and alarm, burglar alarm, etc.) need to be integrated in smart homes. Due to the development of modern digital technology, a wide variety of safety equipment such as security cameras,

smart locks, motion detectors, video doorbells, etc. are constantly emerging on the market, which all help to enhance smart home security systems.

Because of the constant improvement of the standards of living, residents have a growing demand for a good entertainment experience in their smart home. With the video-on-demand service increasing in popularity, people may now get instant access to video or radio content of their own choice, instead of having to passively accept the pre-scheduled programs. This greatly improves convenience. Furthermore, thanks to the increasing number of smart entertainment devices, such as smart speakers and multi-room systems, movies, series and music can not only be voice-controlled but also streamed in different rooms of the building by tracking the location of the residents, so that an immersive smart entertainment experience can be achieved for the residents.

In recent years, with the increasing share of renewable energy sources being integrated into the power grid, the power supply is no longer as stable as before, but only partially controllable and decentralized. This requires a more flexible demand in order to even out the fluctuating supply in the power system. At the same time, the electricity tariff is transiting from 'single rate' to 'time-of-use tariff'. For these reasons, energy management, as a new dimension, is being integrated into smart home functions. The Home Energy Management System (HEMS) is an application that needs to be deployed in smart homes so as to monitor and optimize in-home energy consumption and generation by considering all kinds of factors, including the residents' needs and preferences, the devices' limits and potentials, external pricing and load signals, etc. Figure 2.3 shows an overall architecture of a representative home energy management system [109], which includes five major functionalities: monitoring, logging, control, alarm and management. The appliances in smart homes can be classified into two types, non-schedulable appliances and schedulable appliances, depending on their degrees of freedom. The degrees of freedom of devices refer to devices' flexibility to re-

scheduling their work-item. Devices like televisions, lights, microwaves, etc. have poor degrees of freedom. They need to be working whenever residents want to use them. Therefore they belong to non-schedulable appliances. On the other hand, appliances, such as washing machines and water heaters, belong to schedulable appliances since their work-item can be rescheduled on the premise of respecting the needs of residents. The HEMS can make use of degrees of freedom of schedulable appliances in order to optimise in-home energy use and, in this way, also accomplish the residents' goal of saving money. It is worth noting that the authors in [109] classified the refrigerator to be non-schedulable appliance, which can be explained from the point of view of human control. To maintain food quality, the inner temperature of the refrigerator has to be kept between a minimal temperature and a maximal temperature. The working schedule of normal refrigerators is usually independent of outside control. As appliances become more and more intelligent and controllable, the working periods of refrigerators will be able to be shifted from an expensive pricing time to a cheap one, within a certain range of temperatures. In this case, the refrigerator would then be considered to be a schedulable appliance.

With the advent of smart meters which enable billings based on variable electricity tariffs, the power consumption in the building can be real-time visualized for residents. This, on the one hand, increases the transparency of billing. On the other hand, together with time- of-use electricity tariffs, it provides residents with incentives for load shifting. Benefiting from smart meters, the HEMS can quickly be informed of peak consumption in smart homes and subsequently take corresponding countermeasures. To connect devices in the home seamlessly into an overall smart metering system, a Home Area Network (HAN), as the backbone of the communication between smart meter and home appliances, is needed. Figure 2.4 shows the sketch of the HEMS, including the HAN and the smart meter. The HEMS in the central position on the one hand interfaces with the smart meter to

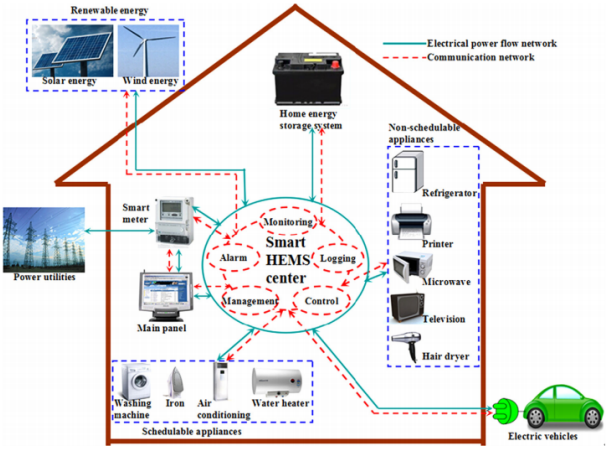


Figure 2.3: Overall architecture of a representative home energy management system [109]

get in-home energy consumption data, and on the other hand communicates with home appliances in the HAN and controls them in different ways.

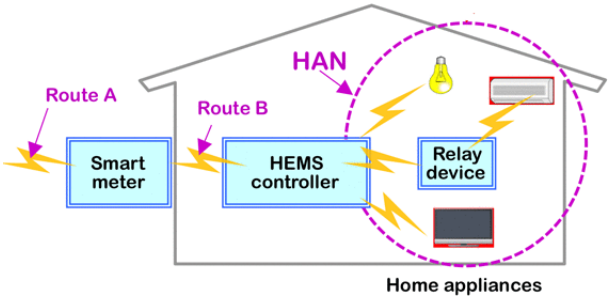


Figure 2.4: Outline of HEMS including HAN and smart meter [35]

Since smart home interconnection specifications and communication technologies are relatively new and still under development, most available communication protocols were developed prior to the advent of the smart home vision [84]. As a result, instead of having one unified communication

protocol for the HAN, so far a dozen of HAN communication and network technologies, which include wired protocols such as HomePlug, X10, Insteon. etc. as well as wireless protocols, such as Z-Wave, EnOcean, ZigBee, etc. coexist on the market. These different kinds of heterogeneous protocols are a challenge for the HEMS to become independent from its underlying protocols. A potential solution for addressing this challenge, is to integrate a protocol conversion middleware which can translate and convert protocol related information to the standard formats used by upper layer services in the HEMS controller, and vice versa. In this way, information exchange and interoperability with HAN devices using various protocols can be achieved.

As more and more roofs are installed with photovoltaic panels, residents are able to produce electricity from renewable sources themselves, rather than merely consuming electricity from the power grid as before. This turns the residents from consumers into "prosumers". The use of electricity from local generation is handled by the HEMS in such a way so as to help the residents to improve their benefits. It does this by maximizing the use of locally generated electricity in smart homes. Because of this, an in-home load management is needed to balance the local power consumption and the local power generation. An efficient device that can be made use of to achieve this goal is the Combined Heat and Power plant (CHP), which offers the capability of co-generating power and heat while consuming a variety of fuels. This device can be extended by installing an electricity-driven heating cartridge into its water tank so that electricity can also be used as another source for generating heat. When the power consumption in smart homes exceeds the local power generation, a CHP can be used to generate power by having it consume gas or other fuels so that the self-supply will be improved. In the opposite, if there is a surplus in the local power generation, a CHP can be used to consume the surplus electricity to heat its water tank, instead of using gas, so that the self-consumption will be improved. In this case,

the CHP is acting as a bivalent system which helps to convert power to gas in a virtual way. Additionally, the HEMS can also adjust the working schedules of the schedulable appliances in smart homes to respond to the variable amount of electricity from renewable sources.

In addition to load shifts of household appliances and virtual conversion of power to gas in bivalent systems (e.g. CHP), the surplus electricity generated from renewable sources in residents' smart home can be dealt with in different ways according to the optimization strategy of the HEMS.

1.) The electricity can be directly fed into the power grid through an inverter and, in return, the home owner will get a credit or a payment for this electricity. Usually the feed-in tariff varies from one kind of renewable to another. Right now, the feed-in tariffs of renewable energy are usually lower than the price of retail electricity purchased from the power grid. So compared to feeding the locally generated electricity into the power grid, it is, at the moment, more cost-efficient to use locally generated electricity to cover the daily power requirements in households as much as possible.

2.) The surplus electricity can be stored in a thermal storage system. The thermal storage can, for example, be a hot water boiler, which provides daily-used hot water for residents in smart homes. Instead of being fed into the power grid, the surplus electricity can thus be used to heat the water in the water tank, and, in this way, the electricity can be stored in the form of thermal energy.

3.) A stationary battery can be installed in smart homes to provide capacity for power storage. When there is surplus electricity from renewables or when the external electricity price is low, the stationary battery in a smart home can store some amount of electricity which later can be used for powering the smart home, especially at peak demand times. So far, the common stationary batteries such as lead-acid batteries and lithium-ion batteries, have not been widely implemented in households yet, because of

various factors like high purchase prices, relatively short service life, high maintenance costs, etc.

4.) Taking into account rising oil prices and the need to reduce greenhouse gas emissions, many countries in the world are starting to advocate and promoting electric vehicles. Since electric vehicles are equipped with rechargeable batteries. They can be integrated into smart homes and function as flexible load to help balance power demand and supply. According to a mobility survey from the European Commission [93], the average daily travel time of cars in many European countries (e.g. France, German, Italy, etc.) is less than 1.5 hours, which means that, most of the time, cars are in the state of parking. In addition to this, the capacity of batteries in electric vehicles is increasing. All of these factors give electric vehicles a huge potential for high flexibility in power storage and load shifting. While electric vehicles can be used for flexible power storage, the residents' demands for their services need to be respected, which means that the travel plans of residents may not be affected. This is the limitation that electric vehicles have, compared to stationary batteries and this limitation will need be taken care of by the HEMS.

The in-home energy use can be optimized in two ways. On the one hand, the HEMS can optimise it, by either backwardly or forwardly shifting load profiles of certain devices with degrees of freedom. On the other hand, residents in smart homes can also be involved in the optimization, by consciously changing their behavior patterns when using devices in their everyday life. Therefore, from the point of view of the residents, it is able to improve energy efficiency and reduce the home's energy costs, and from the point of view of power grids, it can contribute toward the realization of smart grids, by dynamically responding to the imbalance between electricity production and demand introduced by renewable energy. To this end, smart homes need to interface with external entities such as the Distribution Service Operator (DSO) and energy utilities (cf. Figure 2.5), in order to obtain the real-time

feedback information from the power grid. Smart meters, which enable two-way communication with the metering system operator, can receive real-time updated tariff information via their Advanced Metering Infrastructure (AMI). Besides this, the HEMS can obtain external incentives and demand respond signals, such as load limit signals from the utility or other service providers in the power system, via certain HAN interfaces so that the in-home energy use can be adjusted automatically or manually to achieve the goal of demand response of smart grid.

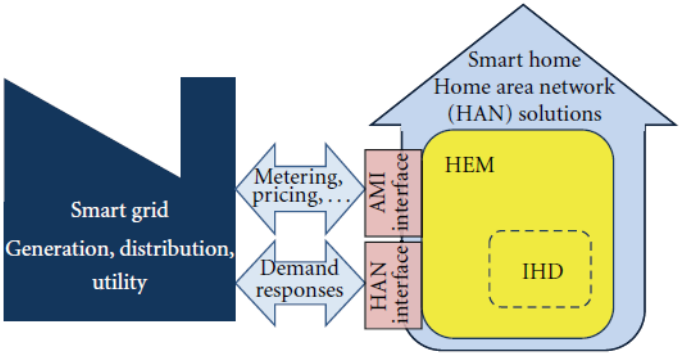


Figure 2.5: Realizing smart grids in smart homes [75]

According to the Smart Home Report 2017, provided by Statista [26], the smart home energy management system market looks promising. Global revenues of the smart home energy management segment in 2017 comprised 2.83 billion US dollars, which is expected to increase by 27.5% in 2022. There are two major reasons for customers to purchase smart home energy management solutions, namely, energy and cost savings, and additional comfort. Thus far, research and products related to smart homes have mainly been focusing on energy conservation. The expected trends of future energy management smart homes include a shift towards local energy production, and storage and integration of electric vehicles.

2.2 Building Operating Systems

In view of all the benefits that the smart home can bring, and the rapid development of information and communication technology, more and more household buildings in the future will realize digitalization and intelligentization. However, smart home technology, is, at present, still in its primary stage and many challenges remain to be addressed. Examples of difficulties that smart homes, are currently facing, can be seen in many areas, such as, compatibility of device communication standards, data sharing of different appliance brands, and unification of data structures used by various co-existing energy management systems in one building. A number of building operating systems have emerged in order to cope with the challenges. Similar to the computer operating system, which coordinates hardware components and provides services for applications software, the building operating system, on the one hand, manages heterogeneous household devices, sensors, storages and other equipment by means of internet of things (IoT) technologies, and, on the other hand, it either optimizes energy use for a building by using its own solution or provides a universal runtime environment, where energy management applications can be deployed. This section gives a brief introduction of a few currently open-sourced building operating systems which mainly focus on aspects relating to comfort and energy.

2.2.1 Energy Flexibility Platform & Interface (EF-Pi)

The Energy Flexibility Platform and Interface (EF-Pi) [5] has previously been known as the FlexiblePower Application Infrastructure (FPAI). It is a building operating system for smart homes which was developed by Flexiblepower Alliance Network (FAN). The mission of EF-Pi is to provide an energy interoperable framework, which, on the one hand, is able to bring together different kinds of household devices using different communication

protocols, and, on the other hand, enables compatibility of heterogeneous energy management services. In a smart home, the EF-Pi software is installed in a so-called FlexiblePower Homebox (FP Hombox), whose running environment is called FlexiblePower Runtime (FP Runtime). Figure 2.6 shows the scope of EF-Pi in a smart home and the stakeholders associated with EF-Pi.

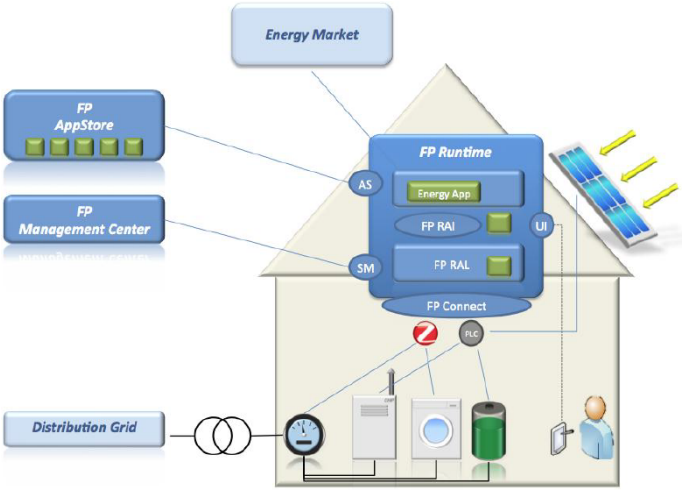


Figure 2.6: The scope of EF-Pi in a smart home and its associated stakeholders [55]

In order to be able to communicate with its ecosystem, the FP Homebox provides various interfaces for different external entities, which include an application store (AS) interface for communicating with the FP AppStore which in turn provides the energy applications for the FP Homebox in a smart home, a service and management (SM) interface for communicating with the FP Management Center which centrally manages all of the FP Homeboxes and is run by an independent party, and an interface for allowing developers to deploy corresponding user interfaces for their applications that are developed based on EF-Pi.

Within a smart home, EF-Pi provides a connect layer to support various communication protocols used by household devices, e.g. Zigbee, Z-wave, PLC, etc. This layer enables the connection between the physical world in a smart home and the EF-Pi environment. On top of the connect layer is a resource abstract layer (RAL) which shields the heterogeneity of the underlying devices by modelling their energy flexibility into unified structures for the upper energy applications. The unified data structure of devices' energy flexibility is named Control Space. In the philosophy of EF-Pi, household devices can be classified into four types, whose corresponding Control Spaces are shown in Table 2.1, according to the characteristics of their energy flexibility. In EF-Pi, it is the devices' energy flexibility, rather than the devices themselves, that are modelled. Because of this, the upper energy applications are independent of the details of household devices since the energy applications only learn about their capability in an abstract form of Control Space, which, in fact, is also their only 'concern'.

Table 2.1: The four Control Spaces in EF-Pi [104]

Control Space	**Description**	**Examples**
Uncontrollable	Has no flexibility, is measurable and may provide forecast.	Photo voltaic, Wind Turbine, TV, indoor lighting, etc.
Time Shiftable	Operation can be shifted in time, has a deadline.	Washing machine, Dishwasher, etc.
Buffer/Storage	Flexible in operation for either production or consumption and operation is bound by a buffer.	Freezer, Heat Pump, CHP, Batteries, EV, etc.
Unconstrained	Flexible in operation for production. The operation is not bound by a buffer.	Gas Generators, Diesel Generator, etc.

The RAL (resource abstract layer) communicates with the upper energy applications by exchanging a number of Energy Flexibility Interface (EFI) messages which are part of the Resource Abstraction Interface (RAI). When a device becomes available to be controlled by an energy application in EF-Pi, it will inform the energy application of its arrival by sending the energy application an EFI message, named ControlSpaceRegistration, via the RAL.

Once the energy flexibility of the device has been changed, an EFI message, named ControlSpaceUpdate, will be sent from the RAL to the energy application. In response, the energy application will send back an Allocation message, which contains instructions on how to use the flexibility described in the ControlSpaceUpdate. Similar to Control Spaces, Allocations generated by different energy applications also have unified and generic data structures. Thanks to the generic Control Space and Allocation, the energy applications can completely decouple from the various household devices that use different protocols and vice versa. This makes the EF-Pi an interoperable platform (cf. Figure 2.7) that can not only integrate all kinds of devices but also can support diverse energy management approaches.

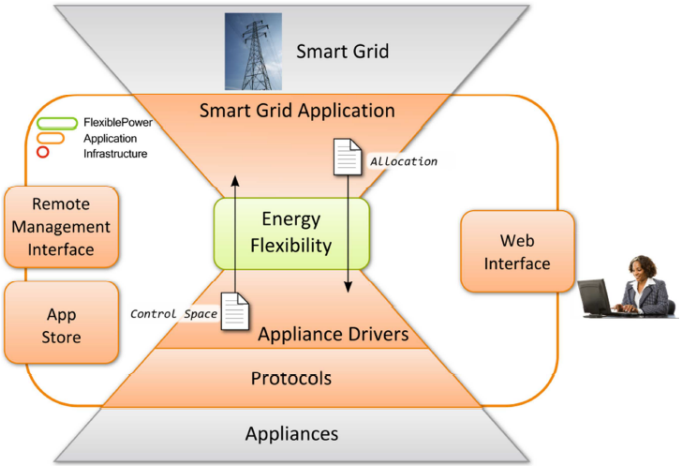


Figure 2.7: Architectural overview of the EF-Pi and the place of Control Spaces and Allocations [28]

In the RAL, each device is usually provided with two components, namely, a device driver, and a device manager, which are used to monitor and control the device. A device driver knows the details of its corresponding device, and exchanges information directly with the device. It is responsible

for reporting the current state of its device and for providing the device-related attribute information, such as commodities that can be produced or consumed by the device, and energy profiles of the device, etc. The device driver does not directly communicate with the energy applications. Instead, it passes data to the device manager which will have further contact with the energy applications. On the one hand, the device manager constructs the Control Space based on the data coming from the device driver, and on the comfort preferences set by the residents, and sends this Control Space to one or more energy applications. On the other hand, it receives the Allocations from the energy applications and translates them into device specific instructions for the device driver which will achieve direct control of the device. In the EIT Digital project HEGRID (Hybrid Energy Grid Management)[3], the μCHP, as a bivalent system, has been integrated into the multi-commodity energy management in smart buildings, on the basis of the EF-Pi framework. To this end, a driver and a manager in the RAL were created for the μCHP. Their state machine diagrams are shown in Figure 2.8, which shows the state transitions and transitional behaviors of the μCHP driver and the μCHP manager.

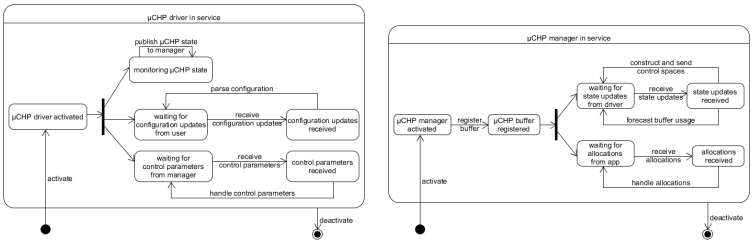


Figure 2.8: The state machine diagram of the μCHP driver and the μCHP manager in the EF-Pi [67]

[3] http://www.aifb.kit.edu/web/HEGRID

In order to achieve data visualization and interaction with the residents, the EF-Pi provides a web-based user interface where residents can manage and view the status of all the installed smart appliances, energy applications and smart grid services. They can also configure and control their devices, and gain more insight into the energy consumption and production inside their building, via this interface [104].

The energy applications can be downloaded from the FP AppStore which is an online repository containing all the applications available for the EF-Pi. So far, a relatively mature energy application that can be integrated into the EF-Pi, is the PowerMatcher[4], which is a smart grid service that aims to achieve demand response based on market mechanisms. Since 2010, the PowerMatcher has been installed in many buildings in different cities of the Netherlands in order to coordinate the energy production and consumption in the buildings.

2.2.2 Open Home Automation Bus (openHAB)

Open Home Automation Bus (openHAB) is an open-sourced home automation platform for integrating different home automation systems, devices and technologies into a single solution. It provides uniform user interfaces, and a common approach to automation rules across the entire system, regardless of the number of manufacturers and sub-systems involved [21]. The developer of openHAB is Kai Kreuzer[5], who released the first version of openHAB in 2010. Since its release, version 1.x of openHAB has been increasingly growing in popularity. To ensure the legitimacy of the use of its open source code, the core framework of openHAB was contributed to the Eclipse Foundation and became the new Eclipse SmartHome framework, based on which a new branch, version 2.x, has been developed. Dozens of

[4] https://fan-ci.sensorlab.tno.nl/builds/powermatcher-documentation/master/html/
[5] http://www.kaikreuzer.de/about/

home automation technologies are currently capable of being integrated into openHAB, e.g. KNX, HomeMatic system, Philips Hue Lighting system, etc.

OpenHAB runs on top of OSGi runtime, which allows modular components (bundles) in a system to be dynamically managed. The core service provided by openHAB is the Event Bus (cf. Figure 2.9) which acts as a communication channel between openHAB services and external systems (e.g. household devices). To enable the interoperability, the external systems connect to the Event Bus via their respective Bindings, which are essentially OSGi bundles and function as adapters, which 'hide' the technical details of the external systems from the openHAB. Bindings constantly update the current states of their underlying devices or systems to the Event Bus and receive commands sent by the openHAB services (e.g. the automation logic or the user interface) via the Event Bus and then translate them into the specific commands to control the corresponding devices or systems.

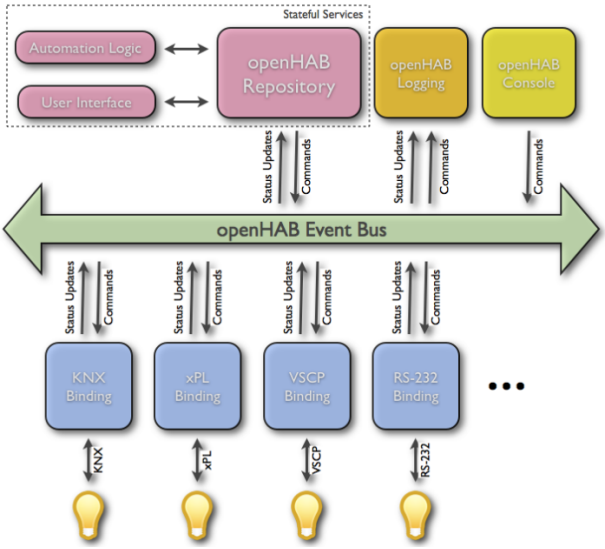


Figure 2.9: The communication mechanism of openHAB [20]

The capabilities of household devices and external services (e.g. weather forecast) are abstracted as so-called Items which are basic data models used by the automation logic and the user interface of openHAB. So far, a set of Item types including Color, Contact, DateTime, Dimmer, Group, Image, Location, Number, Player, Rollershutter, String and Switch are available in openHAB. In the case of complicated conditions, these atomic function units can be organized together to form Group Items which can be displayed as single entries in the user interface of openHAB. One example could be defining a living room as a Group Item to harbour other Number items or device Items such as Dimmers, Rollershutters, etc. The openHAB Repository which connects to the Event Bus, on the one hand, keeps track of the states of all of the Items and, on the other hand, forwards the commands of changing the state of Items from the user interface or the automation logic to the Event Bus.

Besides Items, which are the abstract representations of the functionalities used by the applications, the external physical entities that can be added to the system, such as household devices, web services and sources of information, are abstracted into so-called Things by openHAB. A Thing can potentially provide many functionalities at once, and each of the functionalities can be represented by a specific Channel. Therefore, a Thing is composed of one or more Channels, which on the other side link to Items, so that a Thing can react on events sent for an Item that is linked to one of its Channels, and likewise, it can also actively send out events for Items linked to its Channels [4]. Things connect to the Event Bus via their corresponding Bindings, which could also implement discovery services, so that the Things in a building can be automatically discovered once they are available to the system.

As a generic home automation solution, openHAB allows developers to define Rules for automation processes, e.g. roll down the roller shutter at a certain time in the evening. Rules are texts written in a certain syntax defi-

ned by openHAB. As with Items, Rules are defined in textual configuration files. The Rules and Items can be built by means of the Eclipse SmartHome Designer, which is a special editor provided by openHAB for the purpose of building Rules and Items. So far, Rules in openHAB can only be defined by manually editing text files. A graphical user interface that can replace the textual configuration has not yet been developed. This means, that users are required to have some programming knowledge in order to be able to use the system. The Rules will start working after being triggered. OpenHAB provides users with four kinds of Rule triggers, namely, Item-based triggers, Time-based triggers, System-based triggers and Thing-based triggers, and they can be used to react to different conditions. The interpretation and execution of Rules is taken care of by an integrated rule engine of openHAB.

To enable system configuration and interaction with users, openHAB offers a setup and administration user interface and several user interface alternatives for data visualization and home automation. Except for the administrative user interface, whose content has been pre-determined by openHAB, all other user interfaces can be freely customized to suite the actual situation in different buildings or the different needs of users. This can be achieved by constructing so-called sitemaps which are essentially text files in which the structure of the user interface and the elements that need to be displayed can be defined. A detailed description of these user interfaces and the sitemaps can be found in Chapter 3.

Due to its flexibility, extendibility and customizability, openHAB has increasingly attracted attention and won many awards (e.g. the Duke's Choice Award 2013 at JavaOne and the People's Choice Winner at the post Capes IoT Awards 2014/15) in recent years, which reflects its popularity and influence in the field of home automation. However, just as described on its website, openHAB is not a commercial off-the-shelf product that customers can merely plug in and use. It always requires its users to set up the system before implementation, and many system configurations (e.g. definitions

about Items, Rules and sitemaps) can only be done by editing textual configuration files, which requires certain programming skills. This is a major drawback for laypeople and novice users, who therefore need to be excluded from the target audience of openHAB.

2.2.3 Open Gateway Energy MAnagement (OGEMA)

OGEMA (Open Gateway Energy MAnagement) is a building operating system that provides a common execution environment and a manufacturer- and hardware-independent platform that not only allows to deploy various smart home applications on it, but also supports connections of a variety of devices from different vendors in order to realize building automation and energy management. Its frame architecture is shown in Figure 2.10. OGEMA is being developed in the project "OGEMA 2.0" by three leading Institutes in Germany (Fraunhofer IEE, Fraunhofer IIS and Fraunhofer ISE) within the context of the German and European "Energiewende".

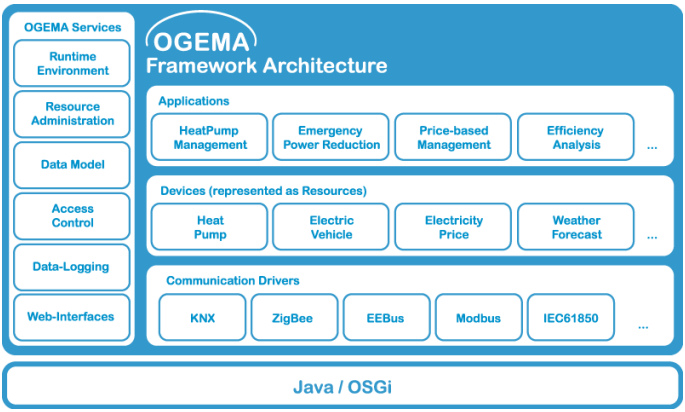


Figure 2.10: Framework architecture of OEGMA [17]

The OGEMA framework is Java-based and runs on top of an OSGi framework, which enables components and services in OGEMA to be dynami-

cally loaded and updated. Within a smart building environment, OGEMA works as an operating system, which on one hand, comes with a set of communication drivers (e.g. drivers for KNX, Z-Wave, etc.) that hide the physical realization of protocol connections so that all kinds of devices from different manufacturers can be integrated into the system, and, on the other hand, provide generic and flexible data models which can be considered as a hardware abstraction layer for various smart building applications deployed on the framework. Besides this, OGEMA offers a variety of services such as resource administration, access control, web interface, etc. for the applications running on it, in order to ensure the security and practicability of the system.

In OGEMA, all kinds of states, parameters and communication data are represented as so-called Resources, and all of the Resources in OGEMA form the nodes of a tree-like structure called the Resource graph, which can represent the current state of the whole system. In spite of having a tree structure, the concept of the Resource graph in OGEMA is different from the traditional resource trees, because a branch of an OGEMA Resource tree can be shared by other OGEMA Resource trees via references, so that this can lead to multiple paths that can possibly be traced to a Resource. When it comes to the location of Resources, paths with references are excluded from the scope of consideration, which means that only the path that does not contain a reference is the location of a Resource. Figure 2.11 shows an exemplary Resource graph in OGEMA where both "MyBuilding/LivingRoom/AirConditioner/ TemperatureSensor" and "MyBuilding/BedRoom/AirConditioner/TemperatureSensor" are paths of the temperature sensor of the air conditioner, when starting from the top-level resource "MyBuilding". However, only the first path is the location of the sensor resource.

The Resources provided by OGEMA can be classified into four categories. They are: Value Resource, which contains actual values, Schedule,

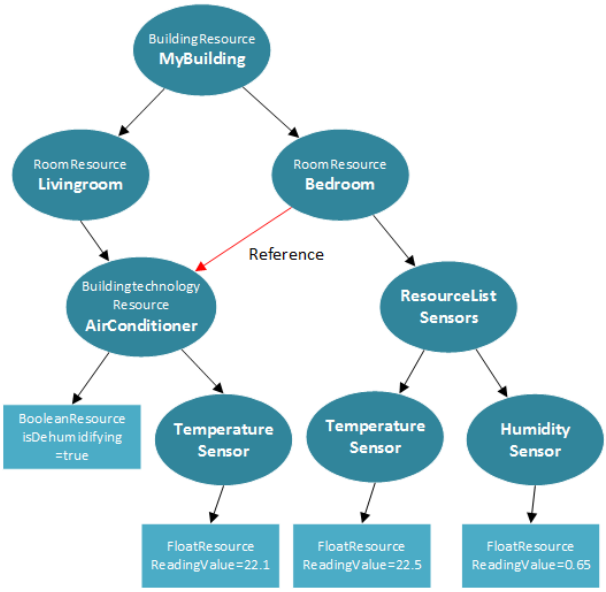


Figure 2.11: An exemplary Resource graph in OEGMA

which represents time series, Resource List, which is a collection of Resources, and Complex Resource, which is made up of some sub-resources and may represent a physical device, a connection, a data container or some configuration settings. The Resources in OGEMA can be monitored by adding different types of event listeners on the Resource graph. For instance, a Resource Demand Listener can be used to monitor the state of a certain Resource type. Once a Resource of the respective type is activated or deactivated, a callback function will be invoked [18]. A Resource Structure Listener informs about the structural change of a certain resource. A Resource Change Listener keeps track of the value or the time series of an individual resource. It is because of all these 'listeners' that the applications deployed in the OGEMA framework can be informed about state changes of

the current system in real time and can then initiate corresponding measures for them.

To ensure the security of the system, OGEMA introduces three different roles, namely, the natural user, the machine user and the framework administrator, in order to implement permissions management. The natural user has access to web resources (e.g. the installed applications) within their permissions via a web user interface. The machine user is mostly used for applications which are assigned certain permissions to perform access to the REpresentational State Transfer (REST) interfaces provided by the OGEMA framework. The framework administrator is an extended form of the natural user who has all of the permissions to create users, view a Resource graph, manage applications and configure the loggers for the OGEMA system. The natural user and the machine user have to be assigned permissions by the framework administrator in order to be able to access the system. The permissions for the two kinds of users are, however, different. For the natural user, the permissions are related to the user interface access, whereas, the permissions assigned to the machine user are about resource access. All the permissions that are defined in the OGEMA framework to control the access to Resources are listed in Table 2.2.

The OGEMA framework comes with a set of abstract data models to describe all kinds of Resources in smart buildings. The most basic data models, from which all other data models are derived, are Configuration, Connection, PhysicalElement and Data. Among them, PhysicalElement is the most important prototype since it can be extended to represent all physical objects in a building, including different kinds of sensors, actors, household devices, rooms and buildings. Inherited from PhysicalElement, a great variety of device models, ranging from simple sensor or actor devices to complex devices, such as white goods, Electric Vehicles and energy storage devices etc., are provided by the OGEMA framework.

Table 2.2: The permissions that are defined in the OGEMA framework [19]

Permission Name	**Description**	**Filters**	**Actions**
Resource Permission	Restrict access of machine users to Resources.	**Path**: The path information of a resource instance. **Type**: The full name of the type definition class of a resource.	**CREATE**: Create a resource in a path with a type. **ADDSUB**: Add a sub resource to a resource. **READ**: Read a resource. **WRITE**: Write a resource. **DELETE**: Delete a resource. **ACTIVITY**: Change the active status of a Resource.
Channel Permission	Restrict access of applications to the low level driver (bus driver).	**Busid**: The address of the local interface the device is connected. **Devaddr**: The address of a device. **Chaddr**: The address of a channel.	**READ**: Read the registers. **WRITE**: Write the registers. **DELETE**: Delete an existing channel.
Admin Permission	Used to distribute administrative rights to the users and application of the system.	No filters.	**USER**: Installing users, and removing them, definition of their access rights. **APP**: Installing applications, restricting their rights, and removing them. **SYSTEM**: Configuring the system settings (e.g. logger settings).
WebAccess Permission	a. Restrict access of applications to the servlets registered by other applications. b. Grant users rights to access registered applications	**Name**: Symbolic name of the bundle associated with the application to be accessed. **Version**: Version or version range of the application.	No actions.

Furthermore, OGEMA is equipped with a flexible web-based user interface framework, where application developers are allowed to design the user

interface for their applications. The user interfaces of installed applications can easily be integrated into the OGEMA web user interface, in the form of widgets. A detailed description of the working mechanism of the OGEMA user interface can be found in Chapter 3.

2.2.4 Organic Smart Home (OSH)

The Organic Smart Home (OSH) is a building operating system that was designed as an organic computer system based on the concept of Organic Computing (OC). OC is a form of nature-inspired computing with a series of life-like (organic) properties. An organic system is a self-organized system that is adaptive and robust and possesses various self-x properties (e.g. self-aware, self-optimizing, self-healing, etc.), and yet maintains its trustworthiness and controllability for a human user. Compared with autonomic computing systems such as the MAPE-K, developed by IBM, organic computing systems are not completely self-organized but rather put emphasis on the interaction with human users and the need for respecting the users' needs. In so doing, the systems are able to adapt robustly to dynamically changing environments without getting out of control. The architecture of organic computing systems is Observer/Controller-based, which is shown in Figure 2.12.

An organic system is usually composed of three essential parts. **SuOC** is the acronym for System under Observation and Control and refers to the complex system that consists of a set of interacting agents/robots/entities. The SuOC can, on the one hand, receive input values from the external environment and, on the other hand, output some values that can be accessed by its outside entities. The system objective is to adapt dynamically to the external changing environment and so maintain some desired behavior. The '**Observer**' is a component that monitors, processes, analyses and predicts the status of the SuOC by looking at certain system parameters. The obser-

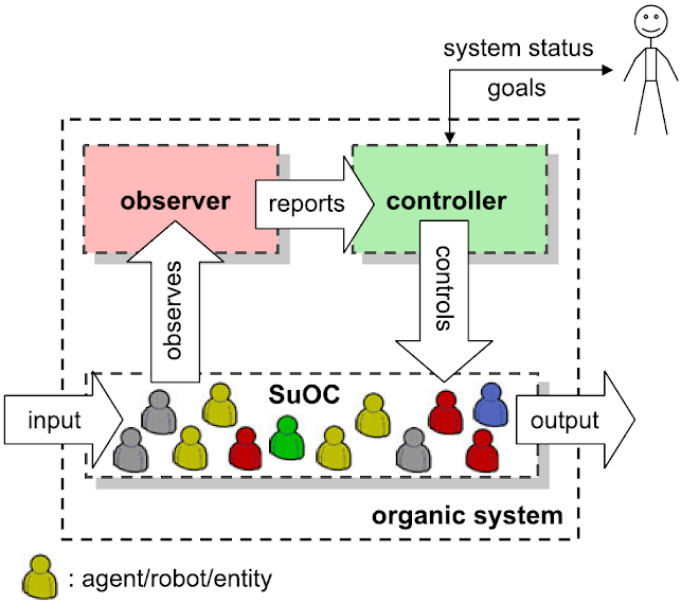


Figure 2.12: Observer/Controller architecture of the organic system [86]

ver reports information about the current and predicted status of the SuOC to the **controller**, which is then responsible for taking the most appropriate action to influence the SuOC in such a way that the desired behavior is achieved or an undesired emergent behavior can be disrupted. The process in which the controller selects the best action that it needs to take in order to control the SuOC, is influenced by the human user, since she is able to observe the status of the SuOC and define system goals which need to be achieved by the controller.

Smart homes that contain a variety of interacting devices and needs, which are to remain under the control of residents, are a typical environment that is well suited to be managed by the solution of organic computing. Thus the OSH is such an organic system, which has been designed to have

a hierarchical Observer/Controller architecture (cf. Figure 2.13). All kinds of in-home appliances, batteries, sensors, smart meters and gateway devices constitute the SuOC whose goals are specified by the owner of smart homes.

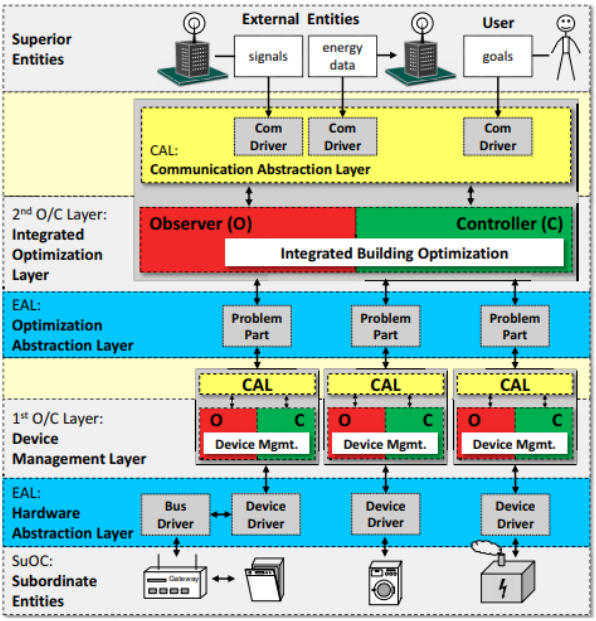


Figure 2.13: Architecture of the Organic Smart Home [83]

Due to the heterogeneity of data formats, interfaces, supported protocols of various devices from different vendors, a hardware abstraction layer which contains corresponding device drivers of the household devices, needs to be added to the SuOC in order to expose the functions of the devices by converting their proprietary data schemes into unified abstractions.

In the architecture of the OSH, there are two Observer/Controller (O/C) layers. The first O/C layer is a device management layer which consists of local O/C-units of in-home devices. Every device is equipped with a local

observer to monitor and characterize its current states and a local controller to receive instructions from the upper layer and control the device by sending commands to the device driver, so as to realize the management of the device status. The second O/C layer is the global optimization layer which consists of a global observer and a global controller. In order to realize unified global optimization, a specific sub-problem is defined for every device, by constructing abstract 'Problem Parts', which can be generically used in the global optimization of the building [39]. Every 'Problem Part' consists of three components, a Bit Vector, an Evaluation Function and a Back Transformation Function. The Bit Vector stores the current solution (e.g. a starting time of a device) which is encoded to a bit string of a specific length. The Evaluation Function returns the expected load profile of the household device, having the instance of the Bit Vector as input. The Back Transformation function re-transforms the abstract Bit Vector into a concrete solution (e.g. a starting time that can be understood by the household device) for the sub-problem. In the following example, a washing machine is used to explain how the Problem Part works in the OSH.

Scenario description: A washing machine has been set by a resident. It is allowed to start running at 2 p.m. and it has to finish the laundry by 5 p.m. The whole washing process will last 30 minutes.

Solution in OSH: The washing machine's degree of freedom at this moment is 150 minutes (2 p.m. - 4:30 p.m.). Assume that the time resolution is one minute, so 8 bits ($2^8 = 256 \geqslant 150$*) are needed for the Bit Vector whose value is the time shifts into the future. Assume that the specific instance of the Bit Vector coming from the optimizer is 00010011 (its corresponding decimal value is 19). By handing the Bit Vector to the Evaluation Function, the Evaluation Function will output the resulting load profile of the washing machine. The Back Transformation Function will calculate the starting time for the washing machine, which is 2:19 p.m. (2 p.m. + 19 minutes).*

After the Problem Part of each device has been constructed by local O/C-units, it is sent to the second O/C layer - the integrated optimization layer, which contains a global O/C-unit consisting of a global observer and a global controller. The global observer assimilates all of the information coming from the local O/C-units and predicts the future system's status. The global controller receives data from the global observer and optimizes the running schedule for all devices in the smart home by learning and controlling the system to achieve the desired behavior.

The global controller realizes the global optimization by using an evolutionary algorithm, whose basic process is shown in Figure 2.14. The starting point of an evolutionary algorithm is to generate an initial population, which is a set of individuals. Each individual is a possible solution encoded in a genotype which, in OSH, is represented by connecting the potential bit vector of each device into the whole. Before starting to optimize, the global O/C-unit randomly generates a group of individuals as the initial population. After that, each individual enters into the evaluation process, which will evaluate the global schedules of household devices, i.e., using the different individuals in the population. During the evaluation, each individual needs to be split up into isolated bit vectors for devices. The load profiles of each device will be generated by inputting every Bit Vector to its corresponding Evaluation Function in the device's Problem Part. The load profiles of all devices will then be combined together so that a global load profile for the whole building will be created. By considering the external signals and the residents' preferences, a so-called fitness value of the individual which represents the quality with respect to the objective function, will be calculated. If the fitness values of all the individuals in the population are not good enough, an evolutionary iteration will be initiated. Firstly, based on a certain selection mechanism, some individuals will be selected as parents for later breeding. In the stage of recombination, one or more offspring will be generated according to different survival schemes. With a certain probability,

some mutations i.e., random changes might occur during recombination between two parents, so that one or more gene values in their offspring will be altered. After recombination and mutation, a new population will be generated. The individuals in this new population will then enter the process of evaluation once more, just like the initial population did. The iteration process will continue until certain stopping criteria are satisfied. This could happen, when a fitness level has been reached, or a maximum number of generations have been reached, or if there have been no improvements over a certain number of iterations. After jumping out of the iteration loop, the global O/C-unit will choose the individual with the highest fitness value as the currently best solution of the whole building.

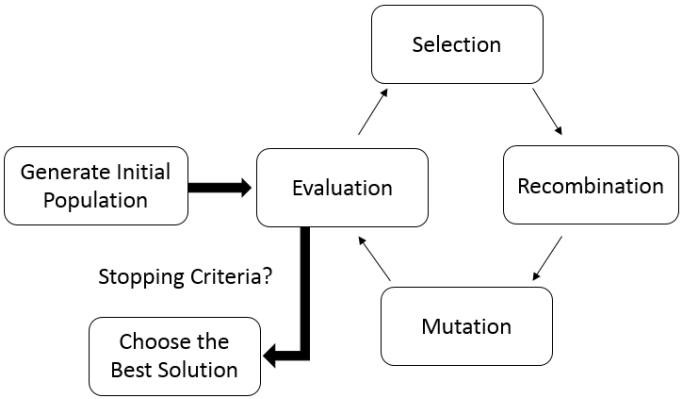


Figure 2.14: The general process of an evolutionary algorithm

The individual generated by the evolutionary algorithm needs to be split up into a number of separated bit vectors for household devices. After being decoded by the back transformation function of the Problem Part of each device, a specific starting time for each device will be generated and sent to its corresponding local O/C-unit, which will further realize the control of the device. Over and above this, the schedule of each device can be interrupted

and rewritten any time by residents according to their wishes, since the OSH puts priority on respecting the needs of the residents in their smart home.

Between the global O/C layer and external entities such as energy utilities and the user, there is a communication abstraction layer which contains a group of communication drivers whose purpose is to decouple the OSH from the external entities.

To date, the OSH has been deployed on two real buildings - the Energy Smart Home Lab[6] at the Karlsruhe Institute of Technology, and the House of Living Labs[7] at the FZI (a research center of computer science). After several years of practice testing, the OSH showed its great potential in providing optimized running schedules for household devices in terms of realizing load shifting and reducing the energy costs.

2.3 A Brief History of Graphical User Interfaces

Since 1962, when the Spacewar, which is the first graphical computer video game, was invented, the evolution of graphical user interfaces has progressed rapidly. It is worth taking a look at the history of user interface design in order to see whether any kinds of trends and patterns of the past might still be helpful in informing today's practice. Unfortunately, software designers who have not understood history and past developments, have often found themselves falling into the same traps and repeating history, but often at much higher costs and lower productivity than the first time around [95]. Just like the continuous development of hardware and Internet technologies, graphical user interfaces have undergone three stages of evolution, namely, desktop user interfaces, web user interfaces and mobile device user interfaces.

[6] https://www.aifb.kit.edu/web/Energy_Smart_Home_Lab
[7] https://www.fzi.de/forschung/fzi-house-of-living-labs/

The programs of computers of the era of batch computing received input information in the form of punched cards. Due to limited computing power, the response time of a normal task was extremely long. As a result, a user interface at that time was considered to be a burden and an unnecessary expense. Until the 1960s, the advent of Command Line Interfaces (CLIs) dramatically improved the situation by reducing the latency from hours to seconds. The interaction model of CLIs was a series of request-response transactions, with requests expressed as textual commands in a specialized vocabulary [95]. However, the disadvantage was, that users needed to remember a large number of commands and parameters in order to be able to complete tasks.

In the 1970s, researchers at Xerox Palo Alto Research Center (Xerox PARC) developed the first graphical user interface based on WIMP (Windows, Icons, Menus, Pointers) paradigm that is still used today. The first personal computer that was equipped with a graphical user interface was named Apple Lisa and was developed by the Apple company in 1983. Apple Lisa introduced drop-down menus which were missing in prior systems (e.g. Xerox Alto). However, the user interface of Apple Lisa did not have color and, lacked the pseudo-3D sculptural effects of modern graphical user interface buttons as well as other impedimenta [95]. The Apple Lisa eventually turned out to be commercially unsuccessful. In the subsequent year, Apple released the Macintosh on the market. Being a major focus of the overall goal of the new product, the graphical user interface of Apple Macintosh has all three main components of a graphical user interface today, i.e., the windowing system, an imaging model, and an API [74]. So, Macintosh had a great impact on the design of subsequent user interfaces. Even today, the Apple Human Interface Guidelines are still recommended reading for anyone desiring to develop good graphical user interfaces [95].

Microsoft tried to release WIMP-based graphical user interfaces since 1985. However, their user interfaces were not successful until Windows

3.1 was released in 1992. Responsiveness to customer feedback, saturation marketing, and the steadily rising resolution of color screens, helped make Windows 3.1 a huge success [95], and from then on, Microsoft released the Windows 9x series, Windows 2000, Windows XP, Windows 7, Windows 8, and Windows 10 in succession, which occupied the market dominance of desktop user interfaces.

By upgrading of the version of operating systems, user interfaces are also constantly evolving to meet the direction of the consumers' needs. In the past two decades, desktop user interfaces have shifted from the initial stage of having only a limited number of black-and-white and crude components, to a transitional stage in which three-dimensional (3D) effects (e.g. shadows, gradients, highlights, etc. in Windows 9x series) and skeuomorphism (e.g. Apple's dashboard design), which mimics the behavior of real-world objects, were popular. However, as the amount of digital information has been expanding and diversifying over the years, the pseudo-3D and skeuomorphism user interfaces were gradually abandoned because of the distraction and cognitive load they brought to users.

As a takeover, the flat design, which is tailored to fit an on-screen experience, as opposed to earlier styles, which were made to mimic a physical experience, has been growing in popularity in recent years [54]. In order to achieve the purpose of rapid transmission of information, the redundant, heavy and complex decorative effects have been removed from the flat design. In terms of design elements, it emphasizes abstraction, minimization and symbolization so that "information" itself can be highlighted as the core for users. Microsoft started to introduce the flat design principles in its release of the operating system Windows 8, which was equipped with a stripped-down user interface that is flat and without frills.

The web user interface normally refers to web pages, which are made up of a set of HTML tags that determine the pages' display in a browser. Its history can be traced back to 1993, when the World Wide Web (WWW)

started growing and the Internet Engineering Task Force (IETF) published a draft of Hypertext Markup Language (HTML). The web pages in those days were simple and static and offered no interaction with users, which mean that, the content delivered by the pages was the same for every visitor. In 1995, the HTML 2.0 specification was published. Web user interfaces at that time already had the strong capability of static display. The year 1997 marks a milestone in the development of HTML, since HTML 3.2 and HTML 4.0 were published in succession as a World Wide Web Consortium (W3C) Recommendation. Powered by the scripting language - JavaScript and Cascading Style Sheets (CSS), the web user interfaces at that time were already able to interact with users to dynamically modify page elements and adjust style properties so as to realize the dynamic HTML (DHTML) pages. Nevertheless, DHTML gradually faded away and, due to the lack of unified standards, was replaced by many different advanced technologies.

As a new version of WWW, the concept of Web 2.0, which emphasizes user-generated content, usability and interoperability for end users, was popularized in 2004 [34]. Since then, WWW has entered a more open and interactive phase in which content can be read and written by users. The Asynchronous JavaScript and XML (AJAX) is an important technology that is associated with Web 2.0. Instead of the synchronized interaction in traditional Web user interfaces, AJAX has implemented asynchronous interaction between browsers and servers with the help of the asynchronous JavaScript. For example, requests generated by a Web page can be sent to the server without reloading the entire page. In 2014, HTML 5 was released as a W3C Recommendation to replace the previous version HTML 4.01 and XHTML 1.0 standards, set up in 1999. Compared to the previous versions, HTML 5 was designed specifically to support multimedia on mobile devices. To this end, new elements and attributes that can change the way in which users interact with documents were introduced into the HTML 5 specification. This plays an important role in the user interface design of recent years, since the

number of mobile devices (e.g. smart phones, tablets, etc.) keeps increasing in people's daily lives.

Unlike traditional computers or laptops, mobile devices usually have small touch-sensitive screens, whose user interfaces, nevertheless, need to transmit the same amount of information to users, as those of computers and laptops. Therefore, the visual style, navigation and layout of content of mobile user interfaces needs to be designed, keeping special considerations in mind. In order to avoid the unnecessary effort of developing multiple versions of user interfaces for different kinds of devices, a new design approach, based on HTML 5 and CSS3, named responsive design, has become popularized recently to create responsive user interfaces, whose layouts can automatically adapt to different screen sizes. Over the last few years, a variety of front-end frameworks which adhere to responsive design standards have emerged, while, at the same time, new ones continue to emerge constantly.

In order to help create user interfaces which have a more consistent look and feel for mobile phones, tablets, desktops and other platforms, Google released a new cross-platform and cross-terminal design language, named Material Design, at the 2014 Google I/O conference. Material Design makes more liberal use of grid-based layouts, responsive animations and transitions, padding, and depth effects such as lighting and shadows [14]. A set of material components can be used to create a consistent development environment for applications and websites across various platforms (e.g. Android, iOS or operating systems of personal computers). So far, Google's Material Design has been implemented in many modern front-end frameworks such as Materialize and AngularJS, which can help to create web user interfaces that comply with the latest web standards.

In conclusion, in terms of appearance, the evolution of graphical user interfaces has gone through a process of moving from primary crudity to medium-term 3D effects and skeuomorphism and then onto today's flat de-

sign. From the point of view of media, graphical user interfaces have moved from desktop to web and, now to mobile devices. Consequently, the design philosophy has also had to change from the previously fixed-size layout, to the responsive design today. All of these evolutionary processes reflect people's ever-changing needs for user interfaces over the different epochs. However, no matter how user interfaces have developed in the past, present or will develop in the future, they all have the same common goal of transmitting information to users. How to deliver information efficiently and effectively to users in a satisfactory manner is always a core issue that needs to be considered by user interface designers.

2.4 Principles for User Interfaces Design

> *"Design principles can guide the designer during the design process and can be used to evaluate and critique prototype design ideas."*
>
> – David Benyon, *Designing Interactive Systems*

Instead of moving directly into user interfaces that are applied to building operating systems only, it makes more sense to firstly focus on basic design principles for general user interfaces, which are applicable to user interfaces for all kinds of systems. In order to ensure that the final user interface can be used by its target users in an effective, efficient and comfortable way, it is indispensable to introduce some basic principles that are to be followed during the design process. Unlike user interface design guidelines, which are more specific and usually intend to guide design details at lower level, e.g. rules for organizing buttons, the design principles laid out in this section consist of more general, fundamental statements that give advice on user interface design issues and problems. Although generally accepted principles for user interface design have not been made available yet, some work has been done

toward that end [103]. For instance, Schneiderman et al. (1987) described the eight golden rules of interface design. Nielsen (1993) developed ten general principles, which are called heuristics, for interaction design. Mandel (1997) proposed his golden rules for user interface design. This section will review these principles as well as other principles which have appeared in different literature and articles on the Internet.

- **Know the target user**

  "When given the choice between obsessing over competitors or customers, always obsess over customers. Start with customers and work backward."

  – Jeff Bezos, *Three Things I Know*

  As implied by the literal meaning, user interfaces are meant to serve users. Successful user interfaces should be user-centered, which means their design should always revolve around the needs of their target users. It is not uncommon to find literature which places emphasis on the importance of users in the process of user interface design. For instance, Dayton et al. (1997) presented a Bridge methodology with three steps, to map the user requirements to the final graphical user interface design. Lewis et al. (1993) came up with a task-centered user interface design, which emphasizes knowing users and their tasks.

  Since the user interface is designed for users, knowing target users including knowing their goals, their expectations, their habits, their skill levels, their limitations, etc. is the most basic principle for designing a good user interface. A versatile and fancy-looking user interface with abundant features, which are, however, not what its target users need, is most certainly of no value. Good interfaces have to provide functionalities to satisfy the users' needs, whilst at the same time ensuring that, the functionalities can be naturally utilized by users

without difficulty or confusion. The best interface is one that is not noticed, and one that permits the user to focus on the information and tasks at hand, instead of on the mechanisms used to present the information and to perform the task [65]. To achieve this goal, what firstly needs to be done, is to know the target users, to then identify the tasks that have to be carried out by different users, and lastly to establish appropriate solutions to allow users to accomplish these tasks as easily as possible.

- **Keep users in control**

  When using a user interface, users feel comfortable and confident if they are able to be in control of the user interface and also get expected responses from it, rather than having to passively accept features etc., which have been pre-set in the user interface. However, it would equally become very cumbersome for users if user interfaces required them to perform every tedious task manually. Thus a trade-off has to be made to ensure that the user interface is smart enough to understand the users' needs and automate some routine tasks so as to help relieve the burden on users as much as possible, whilst, at the same time, still giving users the feeling that the user interface is under their control, by allowing them a satisfactory degree of control freedom.

  One way to keep users in control is to allow them to customize their user interface. Users feel more comfortable and in control of the interface for their desktop if they can personalize it with their favourite colours, patterns, fonts, and background graphics [82]. A good user interface should not force users to have to adapt to what the interface offers, but it should provide users with possibilities of setting their own preferences toward their most natural interaction style.

Configurability could be another feature of the user interface, which allows users to stay in control. Users have different levels of skills as well as different needs. Thus the user interface needs to be flexible so that users will be able to configure what should be displayed in the user interface according to their individual wishes, instead of assuming that 'one fits all'. For instance, novices or basic users, who do not have so much knowledge of the system, may not be interested in or might even feel overwhelmed if the many advanced options which are frequently used by professional users, appeared in the user interface. In this case, users will feel more comfortable if they can configure the user interface by choosing different roles which determine different display modes or directly choose the content to display based on their interests.

To give users more control, Mandel (1997), in his book "The Elements of User Interface Design", recommended allowing users to directly manipulate interface objects. Wherever possible, encourage users to directly interact with objects on the screen, rather than using indirect methods, such as typing commands or selecting from menus [82]. Schneiderman et al. (2009), in the book "Designing the user interface", gave a comparative overview of five primary interaction styles, including direct manipulation, menu selection, form fill-in, command language and natural language. Each of them has advantages and disadvantages. They can be used alone or combined together in different scenarios or for different target users. For example, direct manipulation is appealing to novices, since tasks can be simplified by directly manipulating their familiar interface objects. However, command languages provide a strong feeling of being in control for frequent users, since they learn the syntax and can often express complex possibilities rapidly [101].

In addition, constant and consistent feedback will also enhance the feeling of control [48]. Users feel clear and confident when they are always informed of what will happen before they start an action, and of what has happened, after they have completed the action, by receiving timely and informative feedback in their familiar and understandable language, instead of in machine or developer language. A good user interface should also be forgiving to mistakes made by users and provide solutions e.g. reversible actions to help users recover from an undesired system state.

- **Prevent errors**

"As much as possible, design the system such that users cannot make serious errors."

– Ben Schneiderman, *Designing the user interface*

Making errors is a part of being human. It is inevitable for users to make various errors in the process of interacting with the user interface. Better than providing solutions to recover the system after errors have been made, the best way to reduce loss brought by errors is to prevent errors from occurring in the first place. Don Norman in his book "The design of everyday things", presented two types of errors: slips and mistakes, which represent unconscious errors caused by carelessness or lack of caution on the behalf of users, and conscious errors, caused by wrong judgements or decisions made by users. Page Laubheimer[8], who is a User Experience Specialist in Nielsen Norman Group[9], wrote a few articles which elaborate on how to prevent user slips and mistakes in the user interface design. The following paragraph will present some strategies, which have been collected from

[8] https://www.nngroup.com/articles/author/page-laubheimer
[9] https://www.nngroup.com

these articles, that can help to prevent both slips and conscious mistakes.

Since slips usually happen accidentally as a result of users giving too little attention to the task at hand, one efficient way of avoiding slips is by including helpful constraints in the user interface, e.g. disabling inappropriate options. In so doing, the users' choices are limited within the range of acceptable values so that unnecessary slips can be avoided. Besides this, providing suggestions is also a helpful way of preventing some slips. For human beings, it is always easier to recognize than to recall information since more cues are included during recognition. Reasonable suggestions on hand e.g. search suggestions or good defaults, provide cues for users so that they are able to recognize the useful information they want, instead of providing bothering to recall everything on their own, which is, on one hand, time and energy consuming, and, on the other hand, easily prone to error in most cases. Using user-friendly forgiving formatting is another way to prevent user slips whenever information needs to be entered by users in a specific format, e.g. formatting phone numbers into a natural style for users to check by adding spaces, parentheses, or hyphens.

To avoid conscious mistakes, Page Laubheimer suggested gathering user data to identify specific gaps between the users' mental models and the designers' mental models[10]. Using some methods such as contextual inquiry, field studies, and ethnographic studies which are able to figure out the users' mental models and expectations, designers could then be made aware of corresponding mistakes and take corresponding measures to prevent these from occurring. Some other strategies, like using standard design conventions, having the design

[10] https://www.nngroup.com/articles/user-mistakes

communicate how it can be used, or showing preview results, are also helpful in avoiding conscious mistakes of users.

There are some more guidelines that can be utilized to prevent both slips and mistakes, including removing memory burdens, which could be done by providing suggestions or default values as aforementioned, requesting confirmation before destructive actions, thus allowing users to double-check their actions, and through supporting undo and warning before errors are made.

Besides these strategies, Schneiderman (2009) suggested using the design technique of completing sequences in order to reduce user errors for those actions, in which a number of steps need to be completed. Compared to asking users to achieve their goal via a step by step action, it is more reasonable and not as error-prone to integrate a sequence of those steps into a single action, provided that it is feasible to combine these actions in the user interface so that the errors which could have occurred in each step can be eliminated.

What is more, when errors inevitably occur, it is always essential for the user interface to provide informative and constructive error messages, which are helpful for users in order to raise the success rates in repairing the errors, to lower future error rates, and to increase subjective satisfaction (Schneiderman, 1982).

- **Minimize memorization**

  The human brain is not always good at memorizing, so that a good user interface should help users to relieve their memory load. This goal can be achieved by splitting it into two sub goals: minimizing short-term memory and minimizing long-term memory.

Users may always be interrupted while in the middle of a process, either by some unexpected incident, e.g. phone call, email, etc. or by another task that they might be working on in parallel. Therefore, it is difficult for users to track and memorize the information in the user interface from the last screen to the next one, and a user interface should thus not force users to do so. Computers are great at storing information, but humans not. Good user interfaces should help users to store necessary information that might be needed at a later stage and automatically transfer this information to the users whenever and wherever the information needs to be entered again to relieve users' short-term memory burden. For example, instead of asking users for the data that they have provided before, the user interface should store the previous data and retrieve it for users, as soon as it is needed again and, in addition to this, provide possibilities for users to modify the previous data. This is such a simple interface principle, but one that is often neglected [82].

User interfaces should also support the long-term memory retrieval of users, by providing them with options which can be recognized rather than leaving the user to recall the information without cues [82]. Budiu (2014) in her article "Memory Recognition and Recall in User Interfaces", analyzed how the human memory works and why recognition is better than recall. To promote recognition in user interfaces, she suggested making information and interface functions visible, intuitive and easy to remember. For instance, providing access to history and previously visited content can help users to quickly remember their previous tasks without the strain of having to recall this information themselves. Visible and intuitive interface elements e.g. buttons and menus in the graphical user interface are able to relieve the burden of user recall, resulting from command-line interfaces. Individual tips appearing at the appropriate position at the right time in

the user interface can be accepted more easily than a long intensive tutorial, which, in most cases, can usually not to be kept in the users' memory for a longer period of time.

Beside the aforementioned principles, Mandel (1997) proposed some other principles to reduce the users' memory load, such as providing visual cues, e.g. different indicators to show users where they are, what they are doing and what they can do next, providing interface shortcuts to shorten the number of keystrokes or mouse actions, and using real-world metaphors to transfer knowledge about how things should look and work, etc.

- **Make the user interface consistent**

  An important goal for designers is a consistent user interface [101]. Consistent user interfaces are easy to learn and to use, because, on one hand, users can apply their previously accumulated knowledge about how to work with the user interface to the new functionalities and, on the other hand, the consistent user interface makes the user confident to explore since users will not worry about being startled by unexpected consequences from the user interface that are contrary to their previous experience. When it comes to interface consistency, both conceptual consistency and physical consistency are important [48].

  Conceptual consistency relates to the consistent conceptual model of the user interface. The conceptual model is a logic model which has nothing to do with the physical implementation of the user interface. It includes the intrinsic objects from the task environment of the user, the relationships among the objects, and the operations possible in order to manipulate the objects[100]. Conceptual consistency ensures the mappings are consistent and the conceptual model remains clear. This involves being consistent both internally within the system and

externally as the system relates to its outside environment (Benyon, 2014). Satzinger (1998) did an empirical research to test the effects of conceptual consistency across applications on the end user's mental models, by conducting an experiment. Some evidence was found during the experiment that indicates that more accurate mental models might be developed when conceptual models are inconsistent. As a result, the author urged designers not only to simply focus on improving interface consistency but also to attach importance to design guidelines in order to make the user interface easy to learn and use, even at the expense of consistency.

Physical consistency relates to the physical implementation of the user interface. It can be analysed from two aspects: consistency in presentation and consistency in behaviour. Consistency in presentation means that users should see information and objects in the same logical, visual, or physical way throughout the product [82]. Interface elements like buttons, menus, icons, etc. should have a consistent style, layout, name, color scheme, etc. throughout the whole user interface. Inconsistency in elements such as the positioning of buttons or colors will slow users down by 5-10%, while changes to terminology slow users down by 20-25% [101]. But in some cases, when it is necessary to highlight some elements in order to attract the users' attention, the use of inconsistent elements is very effective (Benyon, 2014 and Schneiderman, 2009). So keeping consistency is not a principle that every designer has to conform to at all times. Violation of this principle could be of advantage in some special cases. Consistency in behavior means that the way an object works is the same everywhere [82]. For example, if one object is clickable and allows users to modify its attributes, then other similar objects or objects, of the same type, should also be provided with the same interactive mode.

During the design phase, designers can follow some design standards in order to ensure that the user interface remains consistent. Existing design standards which are generally accepted, e.g. the behavior of common interface elements like menus, buttons etc. should be followed so that users will be able to transfer their prior knowledge of interaction with the user interface. In addition, detailed guideline documents for specific designs should be developed by designers to enforce consistency (Schneiderman, 2009). To this end, Nielsen (1999) proposed a number of rules for design standards [88], and Schneiderman (2009) presented recommendations for guideline documents and processes.

In the 1940s, at the time when computers had just appeared on the market, the applications were designed not from the user's point of view but from the point of view of the system. That is why they were hard to use. But more recently, increasing attention has been placed on the user, and so, improving the usability of the user interface has become a major concern for designers. Paying attention to design principles can help sensitize the designer to key aspects of a good design (Benyon, 2014). This section collects and analyses design principles from a number of user interface experts. These principles need to be interpreted and extended during the design process of a specific user interface. More specific design guidelines can further be derived from the principles which are used by designers to determine design details. It is worth noting that trade-offs have to be made in some cases where conflicts exist between principles and product goals, under specific circumstances. Principles are not meant to be followed rigidly, rather they are meant as guiding lights to ensure a sensible interface design (Mandel, 1997).

3 Overview of Related Work

Building operating systems can make normal buildings controllable and smart by knitting together different kinds of actors in the building, e.g. appliances, sensors, smart plugs, meters, etc. into one single system. Within a building operating system, actors can exchange information and cooperate with one another in order to achieve common goals. The user interface plays an essential role in interacting with the users and is an indispensable component of a building operating system, since building operating systems cannot work properly without user involvement. Thus, building operating systems always need to remain under the control of users and provide users with timely information relating to system states, e.g. by providing information about local energy use. Since the goals and functionalities provided by different building operating systems vary, their user interfaces are also very different. A first step toward improving user interfaces would be a careful review and evaluation of state-of-the-art user interfaces for current building operating systems. Therefore, this chapter firstly reviews a number of user interfaces used for current building operating systems and also proposes a collection of use cases related to smart homes. The user interfaces are then further evaluated with respect to the proposed use cases as well as a series of technical characteristics, respectively. Part of the work in this chapter has been published in 2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC) [107].

3.1 User Interfaces for Building Operating Systems

This section reviews user interfaces for a number of currently popular building operating systems and analyses their working mechanisms. Most of the current building operating systems have already been introduced in the previous chapter. The user interfaces evaluated in this section were selected according to the following two criteria: firstly, they are open-sourced, and secondly, their use cases are related to smart homes or home automation.

3.1.1 EF-Pi UI

In order to facilitate configuration and data visualization for its applications, EF-Pi (Energy Flexibility Platform & Interface) provides a widget-based user interface framework, which, due to OSGi, supports modular set-up. For instance, Figure 3.1 shows an exemplified user interface of EF-Pi, including four widgets, which are named 'household performance', 'powermatcher controller', 'smartpv panel' and 'storage device manager'. The user interface framework consists of a number of web-based pages, each of which is a container for widgets with different themes. Widgets can be created freely and customized by developers. Usually, every energy application working on top of the EF-Pi runtime has at least one widget to interact with users. Device drivers may also provide their respective widget to display information regarding the current states of the device. Some third party information providers can also offer their applications, along with their widgets, to display useful information such as weather forecast information. In the exemplified user interface showed in Figure 3.1, the current page is named Dashboard, and consists of four widgets. Other inactive pages are Applications, Apps, AppStore and Settings.

The UI (User Interface) framework of EF-Pi and the page templates together determine the layout of the EF-Pi user interface, which is pre-defined

Figure 3.1: An exemplified user interface with widgets of EF-Pi [36]

and compiled by the EF-Pi core and can therefore not be changed by UI developers. EF-Pi places the total design flexibility of its user interface on widgets which give developers full control and freedom to customize their own content. A widget consists of a user-defined Java class in the back-end which can communicate directly with the EF-Pi runtime, and three front-end resources: an html file, a JavaScript file and a cascading style sheet (CSS) file. The Java class defines attributes that will be displayed to end users on the widget and provides methods to access them. The html file and the CSS file determine how and in which style the data should be displayed. The JavaScript file is the connector between the Java class and the html file. It periodically receives updated data from the EF-Pi runtime by invoking methods defined in the Java class, and this data will be further accessed by the html file. Figure 3.2 shows a screenshot of two widgets which were developed based on an EF-Pi framework in a project named HEGRID[1]. The

[1] http://www.aifb.kit.edu/web/HEGRID/en

two widgets serve as an example to help explain the working mechanism of widgets in the user interface of EF-Pi.

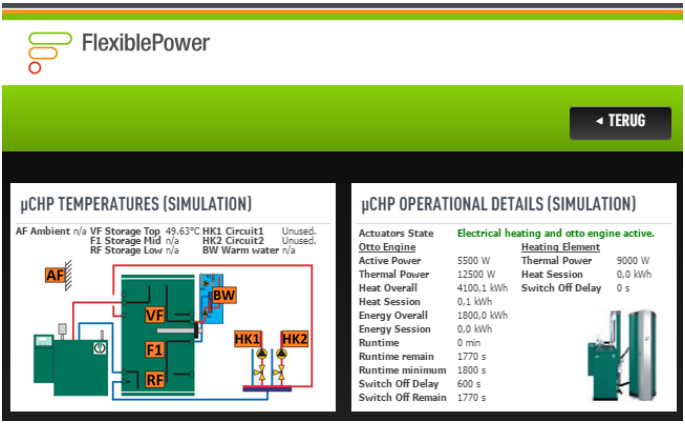


Figure 3.2: Two µCHP driver widgets from the HEGRID project [62]

The two widgets in Figure 3.2 are designed to visualize different parameters and states of a micro Combined Heat and Power Plant (µCHP) in the test-bed of the HEGRID project. One widget is a temperatures widget which shows the temperature at different points in the hot water storage of the µCHP, the outside temperature as well as different circuit temperatures in the test-bed. Another one is an operational details widget which shows the detailed real time parameters (e.g. states, power outputs, etc.) of two actuators of the µCHP - an Otto engine and a heating element. Figure 3.3 uses the example of the µCHP operational details widget to show how the back-end Java class relates to the front-end files of widgets in the EF-Pi and how they work together. The µCHP temperatures widget and widgets from other device drivers or applications share the same working mechanism as the µCHP operational details widget.

In the back-end Java class, there is an inner class, called Update, which contains various attributes of the µCHP, such as current running mode, active

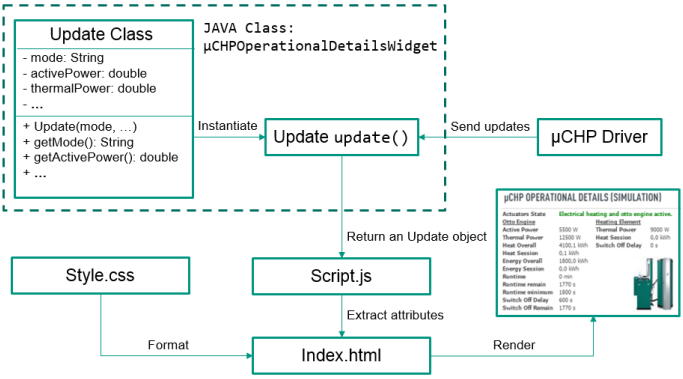


Figure 3.3: The working mechanism of the µCHP operational details widget in EF-Pi

power, thermal power, etc. The attributes defined in the Update class are intended to be displayed on the µCHP widgets. A method named 'update', instantiates an Update object by receiving µCHP's state updates from the µCHP driver and then return the Update instance. With the aid of the technology of Ajax and servlet, the JavaScript file (Script.js) in the web client is able to invoke the update method and get the returned Update instance that contains current states of the µCHP from the EF-Pi runtime. After that, it will extract corresponding attributes from the Update instance and reprocess them by adding appropriate units for the data. The reprocessed data will be accessed by the html file (Index.html), which defines the layout of the widget. Combined with the CSS file (Style.css), the content of the widget will finally be rendered on the EF-Pi UI framework.

The user interface provided by the EF-Pi is simply a framework, which provides possibilities for developers to define their own widgets. It brings high flexibility and customizability, but, at the same time, also weakens the power of the EF-Pi user interface, since the data model provided by the EF-Pi user interface is merely an empty Java class, such as the Update class in

the µCHP operational details widget seen in the example above. Application developers thus need to construct the Java class on their own, which makes every widget proprietary to its underlying device driver or application. In other words, widget developers have to design and develop their widgets from scratch. There are no reusable UI modules provided by the EF-Pi user interface, to help developers simplify or unify their widget development.

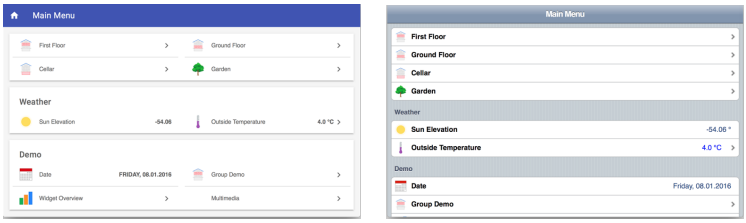
3.1.2 openHAB UI

Currently, openHAB (open Home Automation Bus) provides three kinds of user interfaces in its standard configuration, namely, the Paper UI, the Basic UI and the Classic UI. The Paper UI is an administration tool for configuring openHAB instances. The functionalities supported by the paper UI include add-on management, discovery services of Things, configuration of the system, bindings, services and Things, and setting of preferences. Nevertheless, the Paper UI is not able to cover all of the configuration tasks. For instance, there is no way of editing and managing Items via the Paper UI. As a result, it is still necessary for developers to resort to textural configuration files in some cases.

Other than the Paper UI, which primarily focuses on configuration, the Basic UI and the Classic UI are user interfaces designed for operating the openHAB. These two user interfaces have the same functionalities but different visual effects (cf. Figure 3.4). The Basic UI is the user interface of openHAB 2. Its layout takes advantage of the Material Design Lite[2], which is a front-end template offered by Google. Because of this, the Basic UI not only has a more modern look and feel about it but also has a responsive layout. The Classic UI is the user interface of the early versions (1.x) of openHAB. It is developed on the basis of the WebApp.Net framework and offers the same services as the Basic UI. Compared to the Basic UI, howe-

[2] https://getmdl.io/

ver, the Classic UI has less visual appeal, since its style resembles the style of the old iOS which does not match modern style standards.



(a) The Basic UI of openHAB (b) The Classic UI of openHAB

Figure 3.4: The user interfaces of openHAB [31]

Besides the visual differences, the Basic UI and the Classic UI share the same design philosophy. They use a so-called Sitemap, which is a declarative UI definition that defines the layout of the UI page. A number of UI elements, including Colorpicker, Chart, Frame, Group, Image etc. are available for utilization in Sitemaps. These elements show information or status data of household devices and some of them provide options for interacting with users. The UI elements can be flexibly incorporated into a Sitemap and afterwards rendered to the UI page. OpenHAB specifies a clear syntax for organising the UI elements in a Sitemap. The definitions of Sitemaps are stored in a text configuration file. The operation of this file cannot, as yet, be achieved via the Paper UI. To avoid directly working on a pure text file, openHAB provides an integrated development environment, called Eclipse SmartHome Designer, which supports syntax check and content assist for editing Sitemaps and other text-based system configurations, such as Items and Rules. The following is an example which shows how to define Items and a Sitemap to customize the content on a UI page.

Assume that two switch Items, which represent a light in a kitchen and a dimmer in a bedroom, are defined as follows:

Switch Light_Kitchen "Light in Kitchen" <light> { channel="
zwave:device:bb4d2b80:node30:switch_binary"}

Switch Dimmer_Bedroom "Dimmer in Bedroom" <light> { channel="
zwave:device:73ed5f0c:node2:switch_multilevel"}

The names of these two Items are Light_Kitchen and Dimmer_Bedroom and their corresponding descriptions are "Kitchen Light" and "Bedroom Dimmer", respectively. Both Items use a light as their icons, which are chosen from a default icon set of openHAB. The information in the curly brackets refers to a channel link between an Item and a Thing, which enables the Items to take control of real world home automation devices. Both switch Items are linked to a Z-Wave Thing. After defining the Items, a Sitemap can be defined to specify the structure of the UI page and the elements that will be displayed on the UI page.

```
sitemap demo label="My Demo House" {
    Frame {
        Switch item=Light_Kitchen icon="switch"
        Slider item=Dimmer_Bedroom icon="slider" }
}
```

The name of the Sitemap that is described above is "demo", and "My Demo House" is the title which will be displayed at the top of the main screen of the openHAB's user interface. In the Sitemap, a Switch element which connects to the Item named Light_Kitchen, and a Slider element, which connects to the Item named Dimmer_Bedroom, are defined. It is worth noting that elements in a Sitemap do not always have the same names as the Items that the elements are connecting to. For instance, the dimmer in the above example is defined as a switch Item, whereas it is rendered with a Slider element in the Sitemap since dimmers can be dimmed down or up within a certain range rather than merely two states of 'on' and 'off'.

According to the properties of the two elements, they are rendered in the Sitemap with a switch icon and a slider icon, respectively. The textual description of the elements in the Sitemap are not set which means that they are the same as the descriptions set for their respective Item. Figure 3.5 shows the final result displayed on the Classic UI of openHAB. The layout of the icons, labels, widgets, etc., which finally appear on a user interface, is determined by the openHAB system, therefore the layout cannot be changed by programmers.

Figure 3.5: A demo user interface of openHAB

The Basic UI and the Classic UI of openHAB are implemented in the way of an OSGi bundle. They firstly register a servlet with a Jetty web server which processes incoming requests, and then make use of the Sitemap that has been defined by users, to render corresponding UI pages. This section will now proceed to describe the main components of the bundle of the user interface in more detail. The WebAppServlet is a component, which is responsible for generating corresponding html code based on the Sitemap definition. To this end, a component named SitemapProvider firstly accesses the Sitemap files, loads them and retrieves the one that needs to be parsed. A PageRenderer component then processes the Sitemap page, which is composed of widgets, and constructs the html code by providing the HTML header and skeleton, and delegating the rendering of widgets on the page to the dedicated WidgetRenderers. These then proceed to produce

code snippets for specific widgets. Figure 3.6 shows the entity relationship of these components.

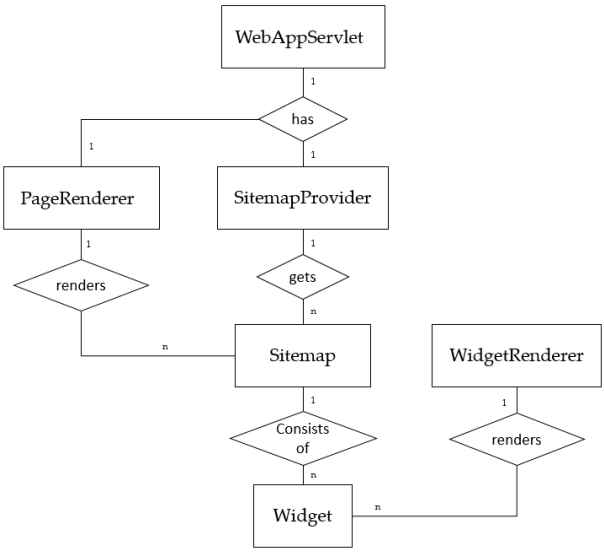


Figure 3.6: Entity relationship diagram of components of the user interface bundle in openHAB

Concluding then, some of the system configurations of openHAB instances can be achieved with the aid of the Paper UI. The direct use of textual configuration files is, however, still needed because of the limited functions provided by the Paper UI. As for the system visualization and control, the Basic UI and the Classic UI are important as they are responsible for parsing the Sitemap file. Thus, to construct a customized user interface for an openHAB instance, developers do not need any front-end programming knowledge apart from mastering the necessary syntax for the Sitemap definition. At the moment, the Basic UI and the Classic UI can only be customized via modifying the Sitemap file, which requires users to know the syntax of the sitemap and to know well the item definitions. In the user interfaces, there

are no visual elements available that allow novices or non-professional users to directly manipulate them to do customizations. In addition to the aforementioned three standard user interfaces, HABmin[3] and HABPanel[4] are also alternative user interfaces for openHAB. HABPanel uses dashboards as content container which consists of different types of widgets. HABmin provides both user and administrative functions. But compared to the Paper UI, it does not support the functionality of installing Add-ons.

3.1.3 OGEMA UI

OGEMA (Open Gateway Energy MAnagement) provides a web-based user interface which is based on AngularJS technology. Similar to the user interface of the EF-Pi, the start page of the OGEMA user interface is also composed of widgets which represent the user interfaces of the installed applications or services in OGEMA. Unlike in the EF-Pi, the widgets on the OGEMA UI page only show simple descriptions of the respective applications. By clicking on any widget, a new web page for the specific application will be opened. OGEMA does not provide any pre-defined data models for the application web pages. They need to be totally taken care of by application developers, which is similar to EF-Pi. Figure 3.7 shows a demo user interface of OGEMA, which appears when running the OGEMA demokit[5].

As described in Section 2.2.3 of the previous chapter, the framework administrator, the natural user and the machine user are three roles allocated in OGEMA. Since the framework administrator is an extension of the natural user, the UI page for the administrator in the user interface of OGEMA is similar to that of the natural user, except that the administrator has three additional options that allow him to do user administration, resource view and logger configuration. The machine users mostly refer to applications,

[3] http://docs.openhab.org/addons/uis/habmin/readme.html
[4] http://docs.openhab.org/addons/uis/habpanel/readme.html
[5] https://www.ogema-source.net/wiki/display/OGEMA/OGEMA+Demokit

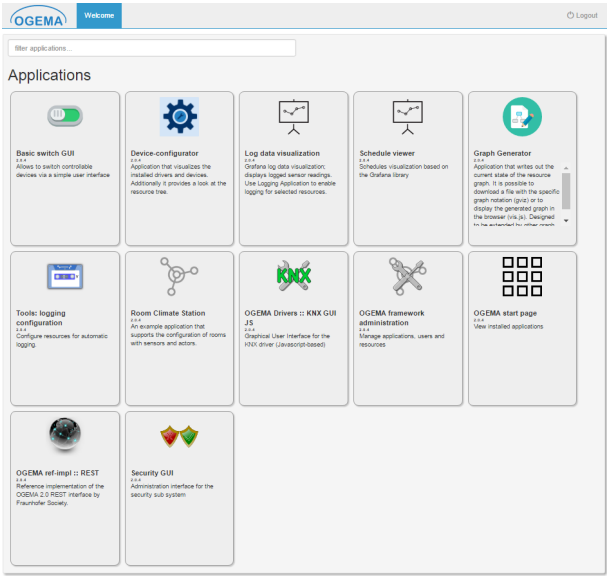


Figure 3.7: A demo user interface of OGEMA

and therefore there is no specific UI page for the role of machine user. Basically, the following components are essential to accomplish the functionalities provided by the OGEMA user interface.

- **FrameworkGUI** implements the application interface. It provides a start method that is called by the framework once this application has been detected, as well as a stop method that is called when the application is stopped by the framework. Within the start method, components of FrameworkGUIController and FrameworkGUIServlet (described below) will be created.
- **FrameworkGUIController** provides a method to get all of the installed applications that the current user has permissions to access and instantiate respective data instances that will be accessed by web resources, based on the properties of those acquired applications. The

installed applications are retrieved with the aid of an OSGi service which keeps track of all installed services.

- **FrameworkGUIServlet** deals with incoming requests from the web client. Particularly, when it receives a request to open the start page of the user interface, it will ask FrameworkGUIController to collect a list of available installed applications and return them to the web client.
- **Web resources** include html files, JavaScript files, cascading style sheets and so on, which can be divided into two categories - framework web resources and application web resources. Framework web resources define the content and style of the start page (cf. Figure 3.7) of the OGEMA user interface. They are pre-defined and compiled in the OGEMA core. Therefore they leave no space for application developers to apply any changes. The application web resources are prepared by application developers, and they can be dynamically loaded with data sources coming from the response of FrameworkGUIServlet.
- **Applications** can be flexibly implemented by application developers. The UI pages of applications can be implemented by defining corresponding web resources. Without web resources, applications will not appear on the user interface of OGEMA. Applications display current system states and provide controllability to users by reading and modifying resource graphs so as to realize the purpose of interacting with the OGEMA framework.

An important aspect for the technical design and implementation of the user interface is the back-end data models for the user interface. For the user interface of OGEMA, the data models in the back-end can be classified into two types: data models for the UI framework and data models for UI pages of applications. As mentioned above, there are no pre-defined data models

for the application UI pages in OGEMA. Application UI pages are totally taken care of by application developers, which is the same as in the case of the EF-Pi. Developers are free to define their own UI data models for their applications if needed.

Unlike the application UI pages, the UI framework of OGEMA already has a number of defined data models for different purposes in its core. Since content displayed on the user interface of OGEMA varies from role to role, UI data models for users with different roles are also different. Natural users are only allowed to view installed applications within their permissions, so data models for natural users are simple. Only two data models, AppsJsonGet and AppsJsonWebResource are needed for showing basic application information on widgets. In contrast, data models for the framework administrator are more diverse, because OGEMA provides administrators with more rights, such as user administration, resource view and logger configuration, except for the basic application management. Table 3.1 lists these data models which are classified according to their supported functions. The definitions of these data models can be found by referring to the source code of OGEMA on its GitHub[6].

To summarise, the working mechanism of the user interface of OGEMA is similar to that of the user interface of EF-Pi. Although a number of data models are available for the UI framework of OGEMA, these data models can, however, not be used by application developers to construct user interfaces for their applications. Instead, OGEMA gives full freedom for application developers to define their own user interfaces tailored to their applications. But at the same time, it has the same disadvantages as the EF-Pi user interface. Since it has no predefined UI data models for the user interface of its applications, which means no modules, that can be reused, are available to application developers.

[6] https://github.com/ogema

Table 3.1: Data models of the UI page for OGEMA administrators

	Application Management	User Administration	Logger Configuration
Data Models	AppsJsonGet AppsJsonAppConditions AppsJsonAppFile AppsJsonAppPermissions AppsJsonAppPolicies AppsJsonAppPolicy AppsJsonGetAppFiles AppsJsonWebResource	UserInformationJsonGet UserJsonGet UserJsonGetList UserJsonChangePassword UserJsonCopyUser UserJsonCreateUser UserJsonDeleteUser UserJsonPost UserJsonAppId UserJsonAppIdList UserJsonCondition UserJsonPermission UserJsonPermissionCondition UserJsonPermittedApps UserJsonPoliciesList UserJsonResourcePolicy UserJsonResourcePolicyList	LoggerJsonGet LoggerJsonGetList LoggerJsonPost LoggerJsonPostList LoggerJsonSizeResponse

3.1.4 FHEM UI

FHEM (Friendly Home Automation and Energy Measurement)[7] is a central home automation server which supports different home automation hardware systems, e.g. FS20, HomeMatic, etc. In the midst of a fragmented market that is full of incompatible home automation technologies, FHEM provides a platform that can combine hardware from different manufacturers into a complete home automation system. It has built-in interfaces between the FHEM server and various hardware systems. These interfaces are capable of converting FHEM control commands sent to corresponding devices and also converting the radio telegrams from the devices to the FHEM server, so that interactions between actuators and sensors from different systems can be achieved.

[7] https://fhem.de/

A standard built-in user interface called PGM2 (cf. Figure 3.8) is provided by the FHEM installation to provide users with easy access to the FHEM. This is the default user interface of FHEM and implements a simple web server. Since FHEM is controlled via its predefined readable/ASCII commands, PGM2 provides a command field which allows end users to send FHEM commands directly to the back-end system in order to manage or control devices in their household, e.g. defining grouping and turning devices on/off. FHEM provides a great number of commands to support various functionalities, e.g. setting attributes of devices, switching devices, monitoring events, etc. Some of them can also be implemented with the aid of graphical components on the user interface, e.g. buttons and drop-down menus. A link to FHEM's command reference is available in the menu of PGM2.

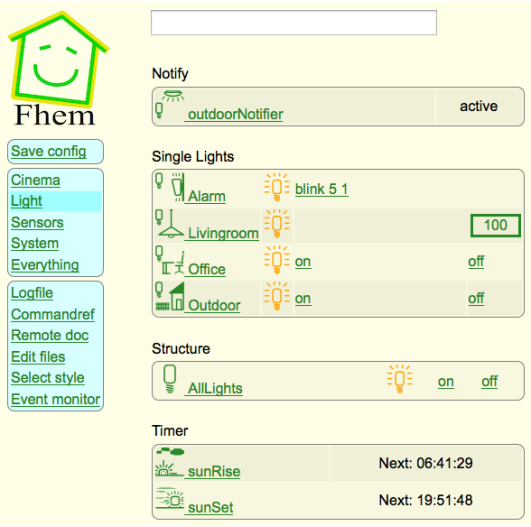


Figure 3.8: The default user interface of FHEM: PGM2 [9]

PGM2 is able to automatically create many radio devices as soon as it receives a message from these devices. The automatically created devices

need to be renamed and this can only be done by entering commands into the command field. The attributes of the added devices can also be edited, either by using commands, or with the aid of graphical components. Any events (e.g. switch on/off) triggered by devices will be recorded and displayed in the event monitor of PGM2. FHEM also comes with a command to support event-based execution. This means the state changes of some devices can trigger the execution of certain events, such as changing the states of other devices. For this purpose, PGM2 has an editor as well as drop down menus which support command editing. However, in order to implement event notifications, users still need to make use of the command field, since there are no available graphical components in PGM2 to support this functionality. Another feature supported by FHEM, is timed switching (i.e., switching devices, not immediately, but at specific time). Similar to event notification, this feature can only be achieved via the direct entry of commands.

In addition, a floor plan module is available in FHEM which supports the placing of icons of switchable devices on the floor plan of a building. A floor plan extension can be integrated into the PGM2 to enable the implementation of this function in the front-end. This floor plan extension allows users to import any background image as the floor plan. Devices that have been added to FHEM can equally be added to the floor plan and can be displayed in one of 9 different styles (cf. Table 3.2). All devices on the floor plan can be dragged and dropped to any places and can also be switched on or off via command tags attached to them. This therefore is supposed to offer a flexible and intuitive way for users to be able to view and manage devices.

An alternative lightweight but feature-rich user interface for PGM2 is called FHEM Tablet UI, which is based on HTML/CSS/JavaScript and thus does not impose any additional requirements on the FHEM server. The Tablet UI is a dashboard which is composed of a number of widgets (cf. Figure 3.9). Compared to PGM2, the Tablet UI is much more flexible for third-party user interface developers to customize their own user interface.

Table 3.2: Display styles of devices in the floor plan of PGM2

Display Style	Description
Style 0	Device icon only (device state)
Style 1	Device name and icon (device name and device state)
Style 2	Device name, icon and commands
Style 3	Device measured value and optional name
Style 4	S300TH temperature and humidity
Style 5	Device icon and commands
Style 6	Device measured value including time stamp and optional name
Style 7	Commands only
Style 8	Device icon and commands popup up

With the help of a drag-and-drop and multi-column jQuery grid plugin, the web page of the Tablet UI can be divided into a number of cells which can be used as 'containers' for organizing widgets according to different topics. At this stage, the FHEM Tablet UI comes with more than 20 types of widgets (e.g. thermostat, switch, label, etc.) in its standard installation.

Figure 3.9: FHEM Tablet UI [10]

Unlike PGM2, which requires users to directly send commands to the FHEM server via a command field, Tablet UI converts FHEM commands into the various attributes of corresponding widgets, which function as the API in interacting with the FHEM server. Although different types of widgets usually have different attributes, there are some attributes which are common to all widgets, such as data-type, data-device and class. In the same way, some actions such as receiving and sending data are also supported by most widgets in the Tablet UI. The corresponding attributes that support these actions, together with the common attributes for all widgets, are listed in Table 3.3.

Table 3.3: Attributes for general widgets of the FHEM Tablet UI [10]

Attributes related to receiving data	**Attributes related to sending data**	**Common attributes for all widgets**
data-get: name of the reading to get from FHEM	data-set: name of the reading to set from FHEM	data-type: the type of the widget
data-get-on: value for ON status to get	data-set-on: value for ON status to set	data-device: FHEM device name
data-get-off: value for OFF status to get	data-set-off: value for OFF status to set	class: CSS classes for look and formatting of the widget

When comparing PGM2 and FHEM Tablet UI, each has its respective strengths and weaknesses. In order to change styles and adjust to terminal devices with various screen sizes, PGM2 offers a list of different styles from which users may select one. Nevertheless, its appearance is relatively crude and it does not look as modern as the FHEM Tablet UI. Since the layout of PGM2 is hard-coded, it offers no flexibilities for UI developers to do customization. However, in terms of the provided functions, it supports comprehensive features for home automation, including event-based execution, timed switching, device grouping, event monitoring, a floor plan, etc. The main challenge here is, that most of these advanced functions can

only be implemented by sending FHEM commands. This places high demands on users to be able to handle these high level commands, and can thus hinder novices or lay users from accessing the system. Compared to PGM2, the dashboard-based FHEM Tablet UI is more flexible, since it allows UI developers to customize the style and content that will be displayed, according to their personal needs. However, the FHEM Tablet UI provides only basic widgets with different attributes, and does not directly support complex home automation activities, such as event-based execution, event monitoring, etc. These functionalities all need to be taken care of by UI developers.

3.1.5 OSH UI

As described in Section 2.2.4 of the previous chapter, the OSH puts special emphasis on the interaction with human users and on respecting the users' needs. This is, of course, also reflected in its user interface. The user interface of the OSH is called the Energy Management Panel (EMP). Currently, there are two versions of EMP available: the KIT EMP [46] and the FZI EMP [45]. These two EMPs are being used in two different buildings. The KIT EMP (cf. Figure 3.10a) is the user interface designed for the KIT Energy Smart Home Lab (ESHL) [7]. The basic hardware setup of the ESHL is presented in [76]. The FZI EMP (cf. Figure 3.10b) has been developed to satisfy the needs of the FZI House of Living Labs (HoLL) [47].

In the KIT ESHL and the FZI HoLL, the communication between the EMP, the OSH and the devices is achieved by means of the Web Application Messaging Protocol (WAMP). WAMP is an open standard WebSocket subprotocol that provides two application messaging patterns, Remote Procedure Calls (RPC) and Publish&Subscribe (PubSub), in one unified protocol [33]. To this end, a router which combines a dealer and a broker to route both calls and events is provided by WAMP. The WAMP router is used as

(a) EMP in KIT ESHL (b) EMP in FZI HoLL

Figure 3.10: The two versions of EMP in KIT ESHL and FZI HoLL [45]

a message bus in both the KIT ESHL and the FZI HoLL to connect the EMP to the OSH and the devices in the building. Figure 3.11 shows how the OSH, the devices and the EMP in the KIT ESHL are connected via a WAMP router and how the information flows between them.

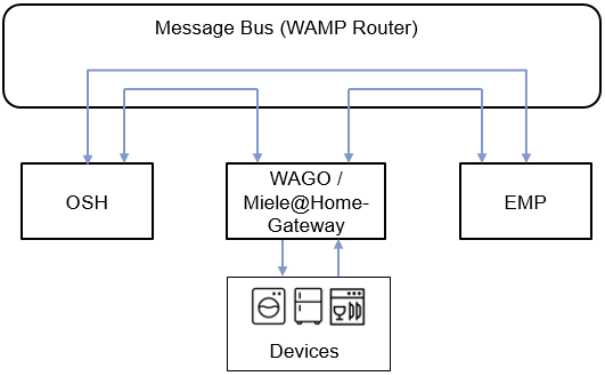


Figure 3.11: Information flows between the EMP, the OSH and the devices in the KIT ESHL

The intelligent devices in the KIT ESHL, such as the washing machine and the dishwasher, are from Miele@home which was developed by the German home appliance manufacturer, Miele, with a view to realizing the

vision of the smart home. The Miele@home appliances in the KIT ESHL are connected to a gateway which can measure the energy consumption of the Miele appliances and publish the corresponding data to the WAMP router where the OSH and the EMP can subscribe topics that they are interested in. On the other hand, the gateway can also forward the commands from the OSH and the EMP to control the appliances. In addition to intelligent appliances, the non-intelligent appliances in the KIT ESHL are connected to a 3-Phase power measurement module from WAGO Corporation, which enables the monitoring and controlling of non-intelligent appliances. Compared to the KIT ESHL, which is a laboratory simulation of a household building, the environment of the FZI HoLL is more complicated, because the FZI HoLL is more than a smart home. It offers additional functional dimensions such as smart automation, smart energy and smart mobility which allow it to incorporate many more devices that cover different types of energy commodities. For this reason, a standardized communication bus called EEBus is used in the FZI HoLL to hide the details of the heterogeneous communication protocols of the various devices.

By using the WAMP router, the EMP can be decoupled from the system details of the OSH. The interaction between the EMP, the OSH and the devices can be realized by registering/calling remote procedures and publishing/subscribing topics via the WAMP router. In both the KIT ESHL and the FZI HoLL, a variety of topics, that expose data from different sources in the system, have been published to the WAMP router. Table 3.4 shows some of the published topics in the KIT ESHL. The EMP and other external systems can subscribe to the topics that they are interested in from the WAMP router, so that afterwards, they will receive the corresponding state updates once the data related to these topics has been updated. The RPC (Remote Procedure Call) supported by the WAMP router, enables the EMP to realize the control over the devices by calling up the remote procedures that have been registered.

Table 3.4: Available topics published on the WAMP router in OSH

Topic Name	**Description**
wizTopic	Readouts of the μCHP, heating cartridge of the μCHP, hybrid storage system and photovoltaic from Wago 3-Phase Power Measurement Module
extendedWizTopic	Readouts of the the whole ESHL from Wago 3-Phase Power Measurement Module
meterTopic	Readouts of the the normal appliances in the ESHL from Wago 3-Phase Power Measurement Module
epsTopic	Energy price signals for the next 24 hours
plsTopic	Energy load limit signals for the next 24 hours
schedulesTopic	Optimization results of the working schedules of appliances
mieleTopic	Readouts of the Miele appliances from Miele home bus
mieleDofTopic	Degree of freedom information for Miele appliances
mieleStartTimesTopic	Start time information for Miele appliances
dachsTopic	Readouts of the the water boiler of Dachs (a μCHP product) from Dachs webservices
tempSensors	The temperature information in the ESHL from BACnet
weatherPredictionTopic	Weather prediction information from OpenWeatherMap service
weatherCurrentTopic	Current weather information from OpenWeatherMap service

The functionalities provided by the EMP in the KIT ESHL deal mainly with the following three aspects: the visualization of energy data, a floor plan based household overview and energy use predictions. For the visualization of energy data, the real-time power consumption and generation in the household, the current energy tariff, external signals from the utility as well as a history of energy use, are displayed. In order to help convey a complete picture of the current energy use in the household, the EMP provides an overview of the energy flows in the ESHL by displaying the energy flow between the ESHL, the power grid and also four high power electrical

units, namely, a photovoltaic cell, a μCHP, an air conditioner and an electric vehicle. By using arrows with different directions as well as frames of different colors in the display, it is easy for a user to obtain a clear picture of the running states of the four electrical units as well as the overall energy condition in the ESHL. The EMP also offers a clickable floor plan for the ESHL so that the status and energy use of each of the devices in the ESHL can be checked individually. So by clicking on different areas in the floor plan, devices located in the corresponding areas, along with their status and power value, will be displayed. These devices can then be further controlled by turning them on or off, or by specifying a degree of freedom for them. In addition to these features, predictions regarding future energy use of certain devices as well as of the entire building are also available in the EMP.

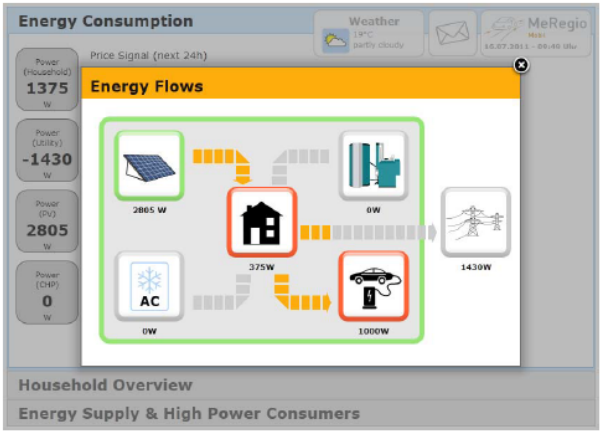


Figure 3.12: The overview of energy flows provided by the KIT EMP [46]

Compared to the EMP in the KIT ESHL, the EMP in the FZI HoLL offers more comprehensive features. For example, its overview of energy flows (cf. Figure 3.10b) is multi-modal, i.e., based on different energy commodities. So besides electricity, it also integrates gas, heat and cold. The floor plan provided by the EMP in the FZI HoLL is configurable and can there-

fore be extended to other buildings beyond FZI HoLL. However, the feature of overview of energy flows cannot automatically adapt to other buildings, when those buildings have different devices from the FZI Holl. This determines that the EMP in the FZI HoLL can only be applied to the FZI Holl. Additionally, some more features, like allowing users to be able to set the thermal degree of freedom for the meeting room with the help of a calendar widget, are also supported by the EMP in the FZI HoLL.

As a a user interface of the OSH, the EMP provides transparent information on household energy consumption and generation, and helps discover and specify degrees of freedom for appliances. Although the EMP in the FZI HoLL offers some improvements on the EMP in the KIT ESHL, there is still more room for further enhancements. For instance, the layout of the EMP is not responsive, and the configuration feature of the EMP is also not well developed. In addition, many features, like overview of energy flows, are proprietary of either the KIT ESHL or the FZI HoLL. They are hard-coded and specifically designed for these two buildings, which limits the flexibility of the EMP in its application to other household buildings.

3.1.6 smartVISU

SmartVISU[8] is an open-sourced UI framework aimed at the visualization of KNX home automation systems. It creates solutions for visualizing KNX-installations by using simple html pages whose content is organized into different types of blocks. Figure 3.13 shows a screenshot of a smartVISU demo. Being a front-end framework, smartVISU can work smoothly with different back-ends, e.g. SmartHome.py, DomotiGa, FHEM, etc. Home automation back-ends have to implement their own drivers in order to integrate smartVISU as their user interface. Since the recommended back-end for

[8] http://www.smartvisu.de/

smartVISU is SmartHome.py[9], the following description of the smartVISU is based on the assumption that SmartHome.py is the back-end system.

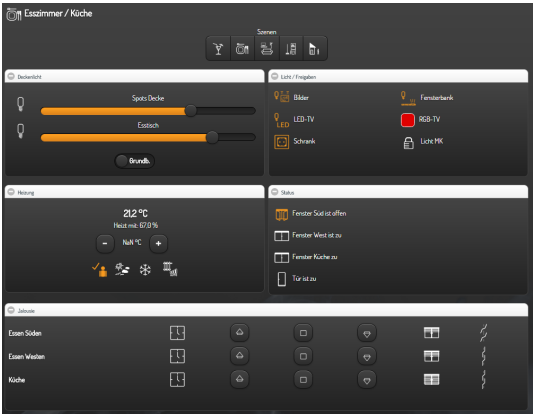


Figure 3.13: The smartVISU demo [27]

In order for the user to be able to visualize different types of device values, home status and other information in a household building, smartVISU provides a collection of widgets which can be classified into the following types: Basic, Calendar, Device, Multimedia, Phone, Plots, Status, Time/Clock and Weather. Syntactically, the widgets in smartVISU are always expressed in a double quoted encapsulated string which contains a widget name and a unique identifier, the "item" or GAD (GAD, short for Group Address, is interchangeable with the term "item" in smartVISU.), which the widget will address, plus some other setting parameters. For example, a button in smartVISU can be expressed in the front-end as follows: {{ basic.button(id, gad, txt, pic, val, type) }}, where basic.button is the name of the widget. The parameters needed by the button widget are: an identifier of the button, an associated GAD/item which will be defined by the back-end system, the text that will be displayed on the button, an icon of the button, a value that will

[9] https://mknx.github.io/smarthome/

be sent out if the button is pressed, and finally, the type of the button's size, which is an enumerated type whose value represents a certain dimension type.

As mentioned above, items (or GADs) play an essential role in widget definitions, since they act as connectors between smartVISU and the back-end system, by specifying different types of KNX addresses within them (e.g. destination addresses for listening and sending information). Besides this, an item also contains the data type, access right, and the Datapoint Type (DPT)[10] of the item. In SmartHome.py, items are defined in a configuration file whose content is organised in a tree-like structure in order to reflect the hierarchical relationship between items. The following is an example of a definition of items in the configuration file.

```
[GroundFloor]
    [[LivingRoom]]
        [[[Spot3Switch]]]
            type = bool
            visu_acl = rw
            knx_dpt = 1
            knx_listen = 0/1/3
            knx_send = 0/0/3
            knx_init = 0/1/3
```

In the configuration file, the number of square brackets determines the depth of corresponding items. An item is addressed according to its path, using dots as separators (e.g. Branch1.Sub-branch_1.1.Sub-branch_1.1.1...). In the aforementioned example, the item "Spot3Switch" is addressed in widgets as "GroundFloor.LivingRoom.Spot3Switch", under which a list of

[10] https://support.knx.org/hc/en-us/articles/115001133744-Datapoint-Type

attributes is specified. According to the attributes in this example, the data type of the item is boolean. The item is allowed to be read and written. The DPT of the item is 1 (bit), which represents a boolean. The item is listening to a KNX group addressed 0/1/3, and it will send commands to another KNX group addressed 0/0/3. It also initializes a KNX group addressed 0/1/3 when SmartHome.py starts up.

Furthermore, in order to facilitate the organization of the widgets which are in same scenes or have similar attributes, smartVISU provides five different design blocks attached to respective html templates which can be utilized directly by developers. Within these blocks, developers are allowed to add any number of devices by simply inserting corresponding widget definitions, as explained above.

Powered by SmartHome.py, smartVISU supports some advanced home automation functionalities. For instance, an automatic timer can be set to perform specific actions, and different scenes can be created to adjust device states in order to match the various needs of the different scenes. These functionalities are all implemented by operating configuration files of the SmartHome.py. Besides this, smartVISU also provides six kinds of widgets to display different charts, including a comfort chart, which indicates comfort zones of relative humidity and effective temperature in a building, and a temperature rose chart which shows the actual and set temperature of every room in one diagram. Furthermore, some configurations of the system, such as configurations concerning the visual appearance, the interface to the back-end, the weather, the phone and the calendar, can also be achieved with smartVISU. Because of its responsive layout, its rich features and its flexible design elements, smartVISU has been implemented in many KNX-enabled home automation systems as the user interface. One aspect that limits the application of smartVISU is the limited variety of widgets available for complex devices. Currently, only five device widgets, including blind, codepad, dimmer, room temperature regulator and shutter are

available. This is because smartVISU was specifically designed for the visualization of KNX home automation systems.

3.1.7 HomeGenie UI

HomeGenie[11] is an open-sourced home automation server that can integrate devices, services and a few of the currently popular communication standards, e.g. Z-Wave, Philips Hue, KNX, etc., into a common home automation environment. For data visualization, it is equipped with a built-in web user interface (cf. Figure 3.14) which is customizable, configurable and extensible. To this end, HomeGenie offers a series of design measures and implementation tools.

Figure 3.14: The demo web user interface of HomeGenie [11]

In HomeGenie, all devices and services are abstracted into so-called modules, whose parameters indicate corresponding features or states of the devices or services. Specifically, each module is identified by a Domain and an Address, a Type and a Widget. The Domain is used to divide all modules into different groups according to protocols or other features. The Address refers to the identifier of a module in its Domain. The Type indicates what

[11] http://www.homegenie.it/

kind of devices or services the module belongs to. Common Types in HomeGenie include Program, Switch, Light, Dimmer, etc. The Widget displays status information and some control options about the module in the user interface.

The widgets in HomeGenie are implemented according to the MVC (Model, View, Controller) design pattern, where View refers to some html code which determines the content that will be displayed on the widget and also what the widget will look like. Model refers to some JavaScript code that will access data from the bound module and render it to the View. Finally, Controller refers to an automation program that, on the one hand, receives commands from the View upon user interaction or other entities and, on the other hand, implements the business logic of the module [12]. In order to facilitate the customization of the system, HomeGenie is equipped with a widget editor to extend its user interface by implementing the first two parts of the design, the Model and View, and with a program editor to implement the third part, i.e., the Controller.

There are already a number of widgets available in HomeGenie. However, the existing widgets can still be customized. Besides this, users are also provided with possibilities to create their own widgets by means of the widget editor and the program editor. HomeGenie utilizes automation programs, which can be coded in the program editor using different programming languages, to implement and extend system functionalities. A multitude of helper classes are available to ease the programming. Each automation program can associate one or multiple modules so that it can display information in the bound widget of the modules and interact with users by adding program options and features to corresponding modules. The added information will be further displayed as option fields in the user interface where users can configure parameters for the programs. In addition, HomeGenie also allows users to create different scenarios (e.g. a scenario for switching group lights on) which can be executed either ma-

nually or automatically to meet different needs in the lives of users. The scenarios can be created by using the function of Record Macro[12], provided by HomeGenie. Through recording macros, all performed commands will be recorded into Wizard scripts, which can either be manually executed by users or be triggered by some time-based events.

The web user interface of HomeGenie is designed as a control panel for the end-users to communicate with the HomeGenie server. It is mainly intended for administration purposes to configure and access all aspects of the system [12]. The user interface consists of a dashboard for displaying various widgets and a number of configuration pages to set up the system (e.g. integrating new devices and installing additional drivers). In addition to the web user interface, which can be used either from a desktop PC or a mobile client, the HomeGenie Plus[13] is another feature rich user interface, which has been especially designed for mobile clients. Because the user interface of HomeGenie is equipped with the widget editor and the program editor, it is flexible and can easily be configured and customized. Nevertheless, since HomeGenie is dedicated to providing solutions for home automation rather than energy management, some services (e.g. setting optimization goals for the building or degrees of freedom for devices) that can help optimize the energy use in a building, are not supported by its user interface. Also, some features that can reflect holistic energy use in a building, such as an overview of energy flows of the building, are equally missing. Furthermore, in order to integrate additional features into the system, users have to be able to develop their own widgets by programming in the widget editor and the program editor which might be applicable to expert users but may exclude novices and laity.

[12] https://genielabs.github.io/HomeGenie/#/docs/scenarios
[13] https://genielabs.github.io/HomeGenie/#/clients

3.2 Use Cases

Due to the lack of unified technical standards, the current market of building operating systems is highly fragmented. A wide range of services related to energy management and home automation is being offered by different building operating systems. Research about defining use cases for building operating systems in different scenarios has been done in some projects. For example, Energy@home[14] has suggested different kinds of use cases for customer energy management systems in residential and commercial buildings. In the FINSENY project[15], a series of different use cases for various building typologies has been defined in one of its deliverables [1]. This section will outline some use cases collected from these reports as well as from the scenarios of the existing building operating systems.

Use Case 1: Basic Home Automation

Devices in the household building can be remotely controlled e.g. switching on or off lights or a washing machine, rolling blinds down or up, etc. via the user interface of the building operating system. The control can be achieved by the intelligent control module embedded in the device or with the aid of smart plugs, if devices are not smart. The reason why this is called basic home automation is that appliances are merely controlled manually by residents in times of need.

Use Case 2: Advanced Home Automation

Advanced home automation is more complex than basic home automation. It allows residents to create different scenarios by executing a series of commands on devices using a single action, or by configuring devices to automatically respond to certain events when some pre-configured conditions are satisfied. For instance, residents in the summer can configure the roller

[14] http://www.energy-home.it/SitePages/Home.aspx
[15] http://www.fi-ppp-finseny.eu/

shutters in their building from the user interface to automatically roll down when the temperature is above a certain value inside the building. Another example could be the time-based control of devices, such as regularly switching on or off lights at a certain time. The major difference between basic and advanced home automation, is that advanced home automation provides residents with possibilities to automate devices in the future rather than having to manually control devices every time there is a need.

Use Case 3: Possibilities to Specify Degrees of Freedom for Devices

The term "degree of freedom" carries different meanings in many fields. When applied to household devices, it refers to the potential of re-scheduling devices. Some devices, like stoves or televisions, do not have degrees of freedom since they have to start working as soon as residents want to use them. Devices like washing machines or hot water boilers have high degrees of freedom since their working schedules can be shifted along the timeline, as long as their own operation constraints can be met. For schedulable devices, users can specify degrees of freedom in the user interface by adding some constraints, such as the required end time of the washing program for their laundry in the washing machine, based on their needs. After specifying degrees of freedom, the building operating system could make use of its optimization algorithms to calculate an optimized working schedule for the devices.

Use Case 4: Visualization of Building-level Energy Data

Residents can view the current states (e.g. voltage, frequency, total power consumption and generation, etc.) or energy flows of the entire building. This use case only refers to the global building-level energy data. It is not about energy data of a single device.

Use Case 5: Visualization of Device-level Energy Data

Residents can view the current state as well as the energy data of every single device, smart plug or sensor in the household. The devices which are viewed in this use case refer to those which are consuming power.

Use Case 6: Visualization of In-house Power Generation

With the increasing popularity of domestic photovoltaic panels, more and more consumers are becoming prosumers. Some buildings are also equipped with μCHP, a bivalent system that can generate heat and electricity at the same time by using different energy sources. Residents can be informed of the power generated by every generating unit in their household.

Use Case 7: Visualization of External Signals

External signals, such as energy cost information or warning messages from the utility company, the demand side manager or other actors, can be accessed by residents. For instance, the EMP of the OSH is not only able to visually display the external signals relating to the current energy tariff and load limits from the utility but also the energy signals for the next 24 hours.

Use Case 8: Role-based Access Control

In order to facilitate security administration and privacy protection, the system access needs to be restricted to authorized users. In so doing, the users that access the user interface of building operating systems can be classified into different roles. Each of them is granted different permissions. Users are only allowed to access the system within the scope of their permissions. For example, the OGEMA system provides three roles including the framework administrator, the natural user and the machine user.

Use Case 9: Floor Plan-based Device Organization

Residents can intuitively view and manage their devices based on a floor plan of a household building. Since there is a visual consistency from the physical world to the floor plan, residents can easily transfer the physical

location of devices in their building to the floor plan. Examples could be the floor plan module in FHEM's default user interface PGM2 and the EMP of the OSH. The floor plan should be flexible and configurable so that it can adapt to different buildings.

Use Case 10: Visualization of Historical Energy Costs

Residents can view the historical energy costs of their entire building. If generating equipment, e.g. photovoltaic panels, are installed, residents can earn money by selling the surplus electricity to the energy supplier. In this case, residents can also view their historical profits from the self-generated electricity. By looking at historical energy costs, residents are able to obtain an overview of the energy use in their household.

Use Case 11: Visualization of Historical Energy Data

Residents can view the historical data of the energy consumption or generation of the devices in their household. Historical energy prices and load limit signals could also be provided together with the energy history of the devices, so that residents can be made aware of whether they have used the devices appropriately the past.

Use Case 12: Prediction of In-house Energy Use

The prediction of energy consumption or generation of devices along with price and load limit signals can be visually displayed in the user interface. However, it would be unnecessary and also impractical to predict energy use for every device in the household. Energy use predictions are worth making for devices which have a high power consumption or generation, such as photovoltaic or µCHP. With this energy prediction information, residents can adjust their power usage to more suitable times in the future in order to either improve the self-consumption/self-supply or shift loads to a cheap tariff period.

Use Case 13: Support for System Configuration

The user interface provides residents with options to configure the underlying building operating system such as integrating new devices, loading additional drivers for specific appliances, adding new users to the system, and configuring parameters for the optimization algorithm used by the building operating system, etc.

Use Case 14: Provision of Value-added Services

Besides the normal services pertaining to home automation and energy management, residents also have access to some other value-added services, e.g. weather information, entertainment, security services, etc. which could help to make daily life more convenient, comfortable and secure.

Use Case 15: Visualization of Historical Data for a Single Resident

It is highly probable that more than one resident resides in one household building. Every resident is able to view their own historical energy data. This use case could be achieved by implementing role-based access control. Each resident can be assigned to a certain role. They might need to be authenticated before viewing their historical energy data in order to protect privacy.

Use Case 16: Integration of Electric Vehicles

The electric vehicle can be integrated into the building operating system and can be used as a flexible mobile energy storage device. The use of electric vehicles always needs to put priority on respecting the residents' actual needs or wishes. To this end, the user interface has to provide options for residents to specify some parameters, e.g. the next use time of the electric vehicle, minimal mileage that the electric vehicle should be able to cover, etc. The building operating system needs to ensure that the battery of the electric vehicle is charged enough whenever residents would like to use it.

Use Case 17: Connection to a User Community

The building operating system of the future could possibly connect to corresponding communities for the purpose of exchanging information, for gamification or for statistic calculation. As a result, residents could view their historical energy data and compare it to the average value of the counterpart in the user community. They may also get suggestions about energy use from residents in the same community.

Use Case 18: Support for Setting Building Optimization Goals

The building operating system allows residents via its user interface to set single or multiple building optimization goals, which could be optimizing for the lowest cost, the maximal self-consumption or self-generation, or the minimal greenhouse gas emission, etc. Different optimization goals can be taken into consideration at the same time, and an optimal solution for the energy use in a building can be provided by using multiple-criteria decision-making methodologies.

In summary, the aforementioned use cases can be regarded as a collection of the available functionalities of the currently popular building operating systems, combined with the suggestions of relevant researchers in the field of smart buildings. On the one hand, they can be used as the implementation reference for the design of the generic user interface for building operating systems. On the other hand, they can also be used as evaluation criteria to evaluate the functional integrity of the user interface for building operating systems.

3.3 Evaluation of User Interfaces for Building Operating Systems

In Section 3.1, a number of popular user interfaces for building operating systems were introduced. These user interfaces provide a wide range of

functions and features which differ from one another in the way they help users getting insight into their in-house energy use, having access to energy data of appliances as well as the whole building, and gaining control over the appliances to make their building smart and energy efficient. This section will evaluate these user interfaces from two angles. One is from the point of view of the use cases presented in Section 3.2. Another is based on the technical characteristics of user interfaces. To this end, a list of technical evaluation criteria is proposed.

3.3.1 Use case based evaluation

This section compares the user interfaces described in Section 3.1 and evaluates them according to the degree of support for the use cases proposed in the last section. The evaluation result can be found in Table 3.5.

EF-Pi and OGEMA only provide a front-end framework for integrating widgets, which has no predefined data models. The content of the widget in the user interface is open to corresponding application developers. Since it makes no sense to evaluate a user interface framework with empty widgets, the demo user interface of OGEMA (cf. Figure 3.7) has been used for the purposes of the evaluation. EF-Pi does not provide a demo user interface, therefore the µCHP widgets which are developed based on EF-Pi (cf. Figure 3.2) have been used in the evaluation. OpenHAB comes with three standard user interfaces, but this section only evaluates two of them, the paper UI and the Basic UI, because the other user interface, the Classic UI, is essentially the same as the Basic UI, but merely looks different. Since FHEM Tablet UI and smartVISU are also just UI frameworks to create visualization, their user demos ([29] and [30]) have equally been used in the evaluation.

The μCHP widgets of EF-Pi are specially designed for displaying energy data of the μCHP, including power generation from an Otto-engine and a heating element. It therefore, only supports use cases 5 and 6.

Table 3.5: Evaluation results of the user interfaces in Section 3.1 based on use cases in Section 3.2

Use Case \ UI	EF-Pi μCHP widgets	openHAB		OGEMA Demo UI	FHEM		OSH		Smart-VISU Demo	Home-Genie UI
		Paper UI	Basic UI		PGM2	Tablet UI	KIT EMP	FZI EMP		
use case 1	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
use case 2	✗	✗	✓	✗	✓	✗	✗	✗	✓	✓
use case 3	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 4	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
use case 5	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
use case 6	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 7	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 8	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
use case 9	✗	✗	✗	✗	✓	✗	✓	✓	✗	✗
use case 10	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
use case 11	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
use case 12	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
use case 13	✗	✓	✗	✓	✓	✗	✗	✓	✓	✓
use case 14	✗	✗	✓	✗	✗	✓	✓	✓	✓	✓
use case 15	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
use case 16	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗
use case 17	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
use case 18	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

The Paper UI of openHAB is not intended for control, but is used to set up and configure the openHAB instance, therefore it only supports use case 13. The Basic UI supports both basic and advanced home automation. The visualization of energy data of devices can be achieved with the aid of power consumption measuring modules. Since the feature of persistence is available in openHAB, solutions for visualizing time series of historical energy data are provided in the Basic UI. In addition, value-added services, such as weather forecasts and Google maps, are also available to be integrated into the user interface.

The demo user interface of OGEMA comes with a few widgets. One of them is a basic switch GUI, which allows residents to view the states of

controllable devices and to switch them. Therefore use case 1 and 5 are supported. Besides this, the user interface also supports role based access control (use case 8) since three roles with different permissions are available in OGEMA. One of the available widgets, called log data visualization, is able to display logged sensor readings, so use case 11 is also supported. Another widget is called OGEMA framework administration. It allows the administrator to configure the system by managing applications, users and resources (use case 13).

Since the built-in user interface of FHEM, called PGM2, provides residents with possibilities to interact with the FHEM server by directly sending commands, both basic and advanced home automation, such as timed switches and event notifications, can be achieved. The Tablet UI, however, does not directly support advanced home automation, since it provides only basic widgets with corresponding attributes. Advanced home automation needs to be taken care of by developers. PGM2 supports use case 9 because it can be extended by a floor plan module. For the visualization of historical log files, PGM2 can directly use the Plot command to create plots, and in Tablet UI, a third party chart widget can be used. What is more, PGM2 supports system configuration by sending commands to the server in the command field. Tablet UI is able to provide value-added services, such as weather forecasts, via integrated or third party widgets.

The two user interfaces, the KIT EMP and the FZI EMP of the Organic Smart Home (OSH), support only basic home automation. These user interfaces allow residents to specify degrees of freedom for appliances. In addition, building-level and device-level energy data as well as the in-house power generation from photovoltaic panels and a μCHP can be visualized in these two user interfaces. External signals such as energy tariff and load limit signals can also be accessed via them. They both have a floor plan to organize the devices in a building, with the difference, that the floor plan provided by the FZI EMP is configurable while the one in the KIT EMP

is not. The historical energy data of devices can also be displayed in both user interfaces. The prediction of in-house energy use is only supported by the KIT EMP. Value-added services and integration of electric vehicles are again supported by both of them. Although OSH is able to optimize in-building energy use, its two user interfaces do not provide residents with options of setting their optimization goals (use case 18).

SmartVISU allows residents to achieve both basic and advanced home automation and to view the energy data of devices. It also provides different kinds of widgets to visualize historical data by creating charts using multiple series. In the demo, historical energy costs, including those for 'yesterday', 'the last seven days' and 'the last thirty days', can be accessed. Use case 13 is supported by smartVISU as well, since it comes with a configuration page providing options for configuring the interface for the back-end, e.g. setting driver, address, port, etc. SmartVISU also provides value-added services, e.g. weather forecast, calendar reminder, caller ID display of the phone system.

The user interface of HomeGenie supports basic home automation as well as advanced home automation, such as creating scenarios. In addition, it is also possible for residents to view both building-level and device-level energy data by creating corresponding modules. Use case 11 is supported by the HomeGenie user interface since it can retrieve the historical data of certain modules with the aid of a Statistics module. The user interface comes with a configuration page, via which residents can configure the system, e.g. enabling control adapters and interfaces, adding groups and modules, etc. Value-added services, such as weather information and security alarm system, are also supported by the HomeGenie user interface.

In conclusion, it can be seen, that none of aforementioned ten user interfaces are able to support all of the use cases proposed in the previous section. Most of them support basic home automation (use case 1), visualization of device-level energy data (use case 5), visualization of historical energy data

(use case 11) and provision of value-added services (use case 14). System configuration (use case 13) is supported by 50% of these user interfaces. Only the demo user interface of OGEMA supports role based access control (use case 8). Only KIT EMP and FZI EMP support specifying degrees of freedom for devices (use case 3), and it is also only these two user interfaces that support integration of electric vehicles (use case 16). Only smartVISU demo supports visualization of historical energy costs (use case 10). Prediction of in-house energy use (use case 12) is supported only by the KIT EMP of Organic Smart Home. Finally, none of the user interfaces supports visualizing historical energy data for the single resident (use case 15), connecting to a user community (use case 17) or setting building optimization goals (use case 18).

3.3.2 Technical characteristic based evaluation

The previous section evaluated popular user interfaces of building operating systems from the point of view of smart home related use cases. In this section, a number of technical characteristics related to successful user interfaces are firstly collected. These characteristics will then be used as criteria for the evaluation of the technical implementation of the user interfaces described in Section 3.1. In order to be able to make an objective evaluation, the criteria selected in this section are those that can be definitely determined, in other words, for which the evaluation result will not depend on the subjective views of different evaluators. For examples, some features related to good user interfaces, like attractiveness or conciseness, are subjective, so that whether the user interfaces satisfy these criteria vary from evaluator to evaluator. For this reason, these subjective features will be excluded from this section.

- **Easy to learn and use**

  Good user interfaces should be quick and easy to learn and use by their target users. This characteristic could be subjective, since users with different skill levels could make different evaluations for the same user interface. To avoid ambiguity, this particular aspect of the user interfaces of Section 3.1 is evaluated, by determining whether professional knowledge of the building operating system in their household is required by users, for the implementation of the use cases of Section 3.2. A user interface is evaluated as 'easy to learn and use' if professional knowledge is not required for residents to achieve their goal, i.e., if users simply need to operate the graphical elements in the user interface.

- **Responsive**

  A user interface is responsive when its layout can fluidly change and respond to fit different screen sizes. This means that it looks good on different types of terminal devices, such as laptops, tablets and phones, by changing its layout dynamically to adapt differently sized screens of devices, e.g. by hiding unnecessary parts. This is an important factor for improving the user experience. The feature helps developers to reduce efforts and costs for user interface development and maintenance, since there is no need for the implementation of different versions of the user interface for differently sized screens.

- **Customizable**

  A customizable user interface allows users to change the appearance and behavior of the user interface according to their preference. When it comes to customization, there is the customization which can be done by the developer, i.e., developer-oriented customization, in which developers have freedom to customize the user interface in terms of its appearance and behavior, e.g. modifying existing compo-

nent templates. However, in this section, this feature will be evaluated according to what extent users, not developers, are allowed to customize the user interface based on their personal needs without having to leave the user interface to do so. Examples could be changing widget features, color schemes, font sizes and so on.

- **Modularized**

  Modularity is the degree to which a system's components may be separated and recombined [15]. Modularized user interfaces consist of a number of loosely coupled and reusable components. Because of this, they are more easily manageable and maintainable than tightly integrated user interfaces, whose components are hard-coded, tightly knitted with each other and specially designed for one specific scenario. Usually the loosely coupled components in modularized user interfaces also provide developers with possibilities for customization. This corresponds to the developer oriented customization which is mentioned but not considered in the previous characteristic of the 'Customizable'.

- **Consistent**

  Consistent user interfaces can help users to improve efficiency and satisfaction and reduce confusion while they seek to complete tasks in the user interface. Making the user interface consistent is also one of the important design principles for the user interfaces, which was outlined in Section 2.4. Consistency can be reflected in different aspects including using the same terminology to represent the same thing, complying with generally accepted conventions, adhering to a consistent style scheme (e.g. layout, color, font and graphical elements) and so on.

- **Multilingual**

  Multilingual User Interface (MUI) originally refers to the Microsoft products that allow users to switch between different languages on a single system [16]. However, the term "multilingual" in this context is not Windows-specific, but applies to user interfaces in general. If a user interface is multilingual, end users can adapt it to various languages at runtime by switching between multiple language options provided by the user interface, without needing to change the source code. In computing, this is referred to as internationalization and localization [13]. This feature is favored by users who prefer different native languages.

The user interfaces in Section 3.1 are now further evaluated in this section, by using the aforementioned technical characteristics as evaluation criteria. The results can be found in Table 3.6.

Table 3.6: Evaluation results of the user interfaces in Section 3.1 based on the technical characteristics

UI / Criteria	EF-Pi µCHP widget	openHAB		OGEMA Demo UI	FHEM		OSH		Smart-VISU Demo	Home-Genie UI
		Paper UI	Basic UI		PGM2	Tablet UI	KIT EMP	FZI EMP		
Easy to learn & use	✓	✓	✗	✓	✗	✓	✓	✓	✓	✗
Responsive	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓
Customizable	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓
Modularized	✗	✗	✓	✓	✗	✓	✗	✗	✓	✓
Consistent	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Multilingual	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

Among the user interfaces of Section 3.1, the Basic UI of openHAB is evaluated as 'not easy to learn and use', since the functionality of advanced home automation provided by openHAB has to be defined via corresponding

rules. To do this, users need know the syntax of the rule definition as well as have knowledge of some system configurations, e.g. Item definitions. FHEM's built-in user interface, PGM2, is also evaluated as 'not easy to learn and use'. The reason for this is that not all functionalities provided by PGM2 can be achieved via simply operating graphical elements in the user interface. Some functionalities used in advance home automation, e.g. event notification, can only be achieved via sending FHEM commands to the server. In this case, users are required to master the professional FHEM commands in order to achieve certain goals. This restricts non-professional or lay users from learning and using the user interface. The user interface of HomeGenie is equally evaluated as 'not easy to learn and use' because users have to resort to a widget editor and a program editor in order to be able to achieve the customization of widgets and features. To this end, users are required to have programming knowledge, such as C#, Javascript, Python and Ruby, which makes it relevant only for professional users.

The Paper UI and the Basic UI of openHAB, smartVISU and the user interface of HomeGenie have a responsive layout. Other user interfaces are either not responsive at all or only partly responsive. For instance, the µCHP widgets of the EF-Pi are not responsive, since the content of widgets is merely visualized via a normal HTML table. FHEM's built-in user interface PGM2 requires users to manually select different styles to adapt to different devices. The KIT EMP of thte OSH is partly responsive. Some of its components, like the chart widget for displaying historical energy use of appliances, can automatically reset scales whenever window sizes change.

Only the built-in user interface PGM2 of FHEM, the smartVISU and the user interface of HomeGenie are customizable. PGM2 provides users with possibilities for selecting different displaying styles. In the configuration page of smartVISU, there are options available to change the visual appearance of smartVISU. With the user interface of HomeGenie, users are able to

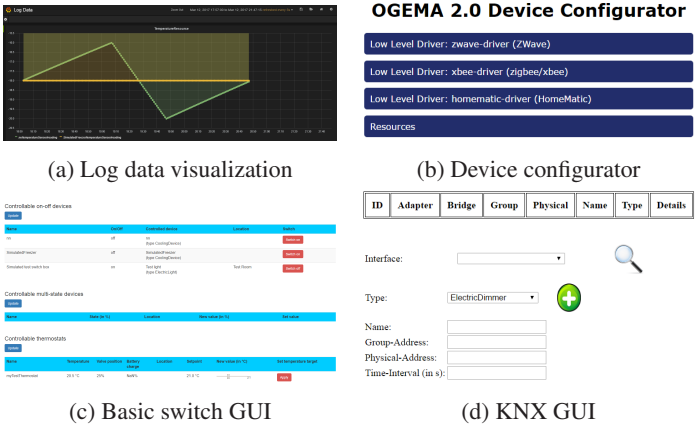
customize existing widgets or add new ones with the aid of a widget editor and a program editor.

The Basic UI of openHAB is modularized because there is a series of predefined Sitemap elements that can be reused by user interface developers. As for OGEMA, its application pages are not modularized, since there are no available data models. Nevertheless, for the role of administrator and natural user, there are some predefined data models (cf. Table 3.1) that can be used for different purposes. FHEM Tablet UI is also modularized because it offers developers a wide variety of widget templates. In a similar way, smartVISU also provides a collection of various widgets that can be reused. In HomeGenie, all devices and services are abstracted into different modules which contain corresponding customizable widgets. There are already a number of widgets available in HomeGenie. Therefore, smartVISU and HomeGenie are also modularized.

All of these user interfaces are consistent, except for the OGEMA demo user interface. The reason why it is evaluated as being 'not consistent' is that the layout styles of the UI pages designed for different applications are very different from one another. For example, Figure 3.15 shows UI pages of four applications in the demo user interface of OGEMA. It is very clear, that the four application user interfaces do not have any visual consistency. They use different background colors, different styles for graphical elements, and different color schemes for highlights.

All of these user interfaces support only one language. There are no possibilities for users to adapt the user interfaces to other languages.

In conclusion, of these ten user interfaces for different building operating systems, none can comply with all of the six technical characteristics which are related to successful user interfaces. In more detail, most of them are consistent and easy to learn and use. Half of them support modularization.



(a) Log data visualization (b) Device configurator

(c) Basic switch GUI (d) KNX GUI

Figure 3.15: Four application user interfaces in OGEMA demo user interface

Four of them have a responsive layout. Only two of them are customizable, and not one provides multilingual support for users.

3.4 Conclusion and Discussion

In recent years, a rising number of local energy management applications and building operating systems have been deployed. Along with these systems there are their respective user interfaces. This chapter is to serve as a reference for a more comprehensive design of the user interface for building operating systems in the future. It started by giving an overview of state-of-the-art user interfaces for different building operating systems. Subsequently, a number of use cases relating to smart buildings were proposed. Finally, the user interfaces were evaluated from the point of view of the smart building related use cases and then with regard to some technical characteristics pertaining to successful user interface design. From the final evaluation results, both strengths and weaknesses of these user interfaces have become evident. None of them can meet all of the evaluation crite-

ria. They all have room for enhancement of varying degrees and in different aspects.

With the future trend of more than one building operating system serving different purposes and operating simultaneously in a building, residents might lose control over their individual targets concerning their building since different building operating systems might interfere with each other. In this case, a common user interface that can be applied to different building operating systems might be required in order to be able to provide a holistic and comprehensive overview of the energy flows as well as ensure a compatible control of various devices. Also, according to findings made, users were more satisfied with the results of the common user interface and performed better with the common interface than with the integrated interface [70]. The common user interface should be generic and flexible enough to interact with users as well as communicate with heterogeneous building operating systems. Thus far, common user interfaces such as these do not yet exist because there are certain difficulties and challenges, but their design should be feasible and it will be of great valued to have them in the future.

4 Design

> Design is not just what it
> looks like and feels like.
> Design is how it works.
>
> *Steve Jobs, Co-founder and*
> *CEO of Apple Inc.,*
> *1955-2011*

With a rising number of competing building operating systems appearing on the future market, it will become increasingly difficult to obtain a holistic and comprehensive overview of the energy flows and devices in a smart home. In a fragmented market, customers could lose control over their individual targets related to their building. This chapter presents the design of a generic user interface for building operating systems, which can be used as a reference for future user interfaces of smart homes to overcome the heterogeneity of different building operating systems and provide various services for residents to increase the comfort and energy efficiency of their building. The design of the user interface can be divided into four aspects - structure, access, content and style (cf. Figure 4.1), which are closely related and complementary to each other. This chapter introduces the design of a generic user interface by mainly focusing on three aspects of the design, namely, on structure, access and content. The style of the user interface will depend on the actual implementation and will be presented in Chapter 5.

Since the concept of a generic user interface for building operating systems is not clear, this chapter begins by, first of all, giving a definition of

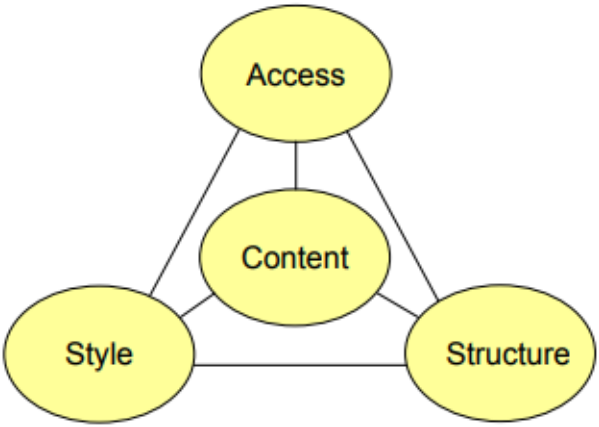


Figure 4.1: Aspects of user interface design [69]

this concept and then presenting a number of system objectives for a generic user interface. After that, an environmental model of a smart home, in which the generic user interface is to be implemented, is introduced, before moving onto the different design aspects of the generic user interface. The aspect of structure, as shown in Figure 4.1, is first focused on in the next section in which an architecture of the user interface, which will make the user interface generic, is proposed. The section that follows, then deals with the aspect of access, as shown in Figure 4.1. Due to the complex environment of a smart home, the users of the generic user interface are classified into different roles. These roles, together with their permissions, are described in this section. After this, a series of generic data models, which are needed by the user interface in order to facilitate the mutual access between the user interface and building operating systems, are introduced. and lastly, the aspect of content, as shown in Figure 4.1, is dealt with at the end of this chapter in which a number of functional components of the generic user interface as well as their relationships are presented. Part of the work in this chapter has been published in the Journal of Energy Informatics [106].

4.1 The Definition

Since the beginning of the 21st century, smart home or home automation technologies have been increasingly becoming popular and changing the way people live. At the same time a variety of smart home related terminologies have also been widely used but not uniformly defined, such as the term 'generic user interface' for building operating systems, which is the focus of the work done in this thesis. As a general term, a user interface has different commonly accepted definitions. More specifically, it can apply to a building operating system, in which case its concept is still self-evident. However, when it comes to the term 'generic user interface' for building operating systems, no definition for this concept has yet been found in the existing publications. For the sake of clarity, the concept therefore needs to be clearly defined in the initial stage of the design. In this thesis, a generic user interface for building operating systems is defined to meet the following requirements:

- **Remote reachability**. In order to enable a user interface to be remotely reachable, it is highly desirable to provide a web-based GUI (Graphical User Interface) system since it is a cross-platform software that is not dependent on any specific hardware or operating system [51]. The traditional desktop user interfaces which need to be installed on a local computer are very platform dependent and require much effort for maintenance and upgrading. The web-based user interfaces, on the other hand, are flexible and can be accessed from any computer which is connected to the Internet via a standard browser. This is the reason for its wide-spread increasing popularity.
- **Responsiveness**. A generic user interface should be responsive. According to a survey[1] conducted by the Pew Research Center in 2015, 68% of Americans are smart phone owners, up from 35% in 2011.

[1] http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/

At the same time, 45% of all adults also own a tablet. This growth has continued throughout the years. With the increasing number of mobile devices of different screen sizes, it is important to create a responsive user interface by using the Responsive Web Design [64], which can ensure that the user interface looks good on different types of devices by adapting the layout to suit different screen sizes, thus delivering a good and consistent experience to users.

- **Configurability**. If a user interface is configurable, some features (e.g. appearance, layouts, displayed content, etc.) of the user interface can be tailored by users according to their needs or preferences. As one of the design principles for user interface design, configurability enables users to realize the personalization of their user interface. This personalization enhances the sense of control of the user and encourages an active role in understanding. It also makes allowance for personal preferences and differences in experience levels, thus leading to a higher user satisfaction [65].
- **Role management**. A generic user interface should be applicable to users who hold different roles. Most designers understand at some level that it is not so much users themselves but the roles that they play in relation to a system that must be taken into account in user interface design [56]. When it comes to the field of smart buildings or smart homes, although usually not too many residents live in a household building, in many cases (e.g. multi-tenancy) household members do have different roles associated with different household practices. By failing to recognize that users value time, roles and relationships in their domestic lives, there are growing concerns that smart home technology could start dominating people, rather than the other way around [105, 59]. A role-based user interface is able to assign users to different roles which have different permissions to access the system. This leads to a higher protection of privacy and a better acceptance by

users. The restricted access to data or certain functionalities ensures confidentiality, integrity, and accountability [68].

- **Flexibility**. A generic user interface should be based on generic data models which are flexible to deal with different building operating systems. A generic user interface is not specifically designed for a particular building operating system, but on the contrary, the architecture of the generic user interface should be loosely coupled with its underlying building operating system, so that it is flexible enough to apply to different household buildings with different kinds of devices.
- **Generality**. A generic user interface should be able to cover extensive use cases relating to the context of the smart home. From the point of view of functionalities, the more use cases that can be covered by the generic user interface, the better. Supporting only limited features will place restrictions on the scope of application of a user interface, i.e., it will affect the applicability and generality of the user interface to a great extent. Chapter 3 reviews the state-of-the-art user interfaces for building operating systems and then proposes a series of common smart home related use cases, which can be used as a reference for the functional design and evaluation of a generic user interface for building operating systems. This set of use cases is referred to in the following chapters when discussing generality.

The aforementioned definition is aimed at eliminating ambiguity and delineating the scope of the work in this thesis. The requirements of a generic user interface for building operating systems listed above, on the one hand, serve to provide guidelines for the design of the generic user interface described later in this chapter, and, on the other hand, they also provide criteria and a basis for the evaluation of the prototype of this generic user interface. The evaluation will be presented in Chapter 6.

4.2 System Objectives

As discussed in Section 2.1 of Chapter 2, the traditional uses of smart homes have been in the areas of comfort, security and health. In recent years, with the increasing awareness of energy conservation, energy, as a new dimension, has been integrated into the smart home functionalities. A services aggregation system that can cover all the aspects could be implemented, but this is not thought to be an optimal solution since users are too diverse and the home is a difficult application context, therefore, any future system should meet these different needs with different solutions, by providing an in-context system that is flexible to meet the wide range of the users' needs [69].

The user interface designed in this chapter cannot provide solutions to solve all the problems in a smart home. Instead, it mainly focuses on the aspects of comfort and energy, since currently the main motivation for consumers to receive smart home technology is to save energy, and to increase the comfort of their homes [50]. Therefore, the major goal of the user interface in this thesis is to help users to improve the comfort as well as the energy efficiency in their building whilst at the same time ensuring good usability. Specifically, the overall goal of the system can be broken down into the following smaller objectives:

- Provide generic data visualization and control for heterogeneous building operating systems in smart homes
- Provide options to help building operating systems to exploit the potential of load shifting in household buildings
- Provide smart home users with not only a holistic but also a clear view of the energy use in their building
- Provide smart home users with as many possibilities as possible to control appliances in their building

- Provide useful reference information to help smart home users to improve their energy awareness so as to achieve the goal of saving money
- Provide a relatively large application scope, which means the user interface is able to meet the different needs of different household buildings
- Provide intuitive displays which allow users to learn to use the user interface quickly and to complete tasks by using the user interface efficiently and effectively

This thesis is limited to the aforementioned system objectives related to comfort and energy. Other aspects related to smart homes, such as healthcare, security and entertainment, do not fall within the scope of this thesis. The corresponding use cases pertaining to these latter aspects include ambient assisted living (e.g. providing assistance for elderly and disabled people), various security alarms (e.g. burglar alarm and smoke alarm) and entertainment systems (e.g. multi-room systems for music), etc. What is more, the user interface designed in this chapter only accepts traditional media (i.e., text, graphics drawings, images) for information transmission. Other media objects such as audio and video are not supported by the user interface.

4.3 Architecture

Current building operating systems are basically equipped with their own proprietary user interface for their application scenarios based on their specific APIs (Application Programming Interfaces). For instance, the Energy Management Panel is the user interface of the OSH. EF-Pi provides a UI framework, which provides some empty widgets that need to be developed by application designers as its user interface. OpenHAB uses the Paper UI

and the Basic UI as user interfaces to realize different goals. These user interfaces are tightly coupled to their underlying building operating systems. Since the smart home market, at this stage, is still highly fragmented, current building operating systems all use different standards and support only their own functionalities, with the result that their user interfaces are heavily coupled with them and can only support a limited number of use cases relating to smart homes or are not flexible to be extended to other household buildings.

Having a generic user interface which can be used for different building operating systems is a solution to deal with the aforementioned problems. Figure 4.2 shows the architecture of the generic user interface designed for this thesis. The key part that makes the user interface generic and extendible is the generic data models, which are appropriate abstractions of objects that are needed by the generic user interface. The details related to the data

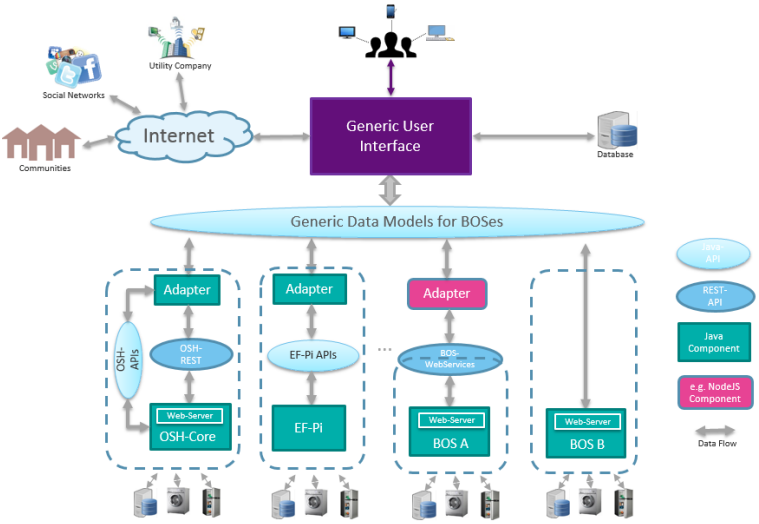


Figure 4.2: Architecture of the generic user interface for building operating systems

models will be explained in Section 4.6. The existing building operating systems (e.g. the OSH, the EF-Pi or another random building operating system named BOS A, which are represented by the first three dotted boxes in Figure 4.2) have their own APIs or web services therefore they need an adapter to convert their proprietary data models to the generic data models so that the generic user interface can apply to them. The adapter component can be located in different places except in the generic user interface. It would be undesirable to do the conversion inside the generic user interface, because in that case, the generic user interface needs to know the APIs or data models of every single building operating system, which is supposed to be neither logical nor realistic. The adapter can be implemented either inside the building operating systems, or as an outside isolated component, such as a NodeJS component, which receives data from a building operating system via web services and then implements the data conversion for the generic user interface. Some future building operating systems might even forego designing their own data models for their user interface. Instead, they could directly benefit from the design presented in this chapter by accessing the generic data models so that they could utilize the generic user interface to interact with users rather than design their own user interfaces from scratch. In so doing, a lot of effort could be saved.

Although building operating systems usually have their own database to store configurations or historical data, the data stored by the building operating system might not meet the needs of the generic user interface. In this case, the generic user interface will, however, still need to access the database in order to realize specific functions. The interaction with the database can be achieved by means of web services.

Furthermore, building operating systems of the future cannot be seen to exist as isolated units, especially in this era of information explosion where information sharing and exchange have become increasingly popular. To begin with, it is very clear that the generic user interface for building ope-

rating systems needs to be able to connect to a utility company in order to not only keep users informed about the status of power supply and demand in the power grid by receiving time-varying electricity prices, but also to provide users with customized services (e.g. reminding users of their billing limits) from the utility. If the energy information from utility companies were available as structured data instead of the usual files (e.g. PDF files), in other words, if the energy data were organized with standard data models, there would be a huge potential for the generic user interface to extract useful data from the information provided by utility companies and reuse the data for providing users with a more comprehensive visualization of their historical energy use or for other purposes.

Furthermore, nowadays more and more people, especially the young generation, desire to share their lives on social media, such as Facebook and Twitter. For this reason, it would be beneficial if the generic user interface would allow users to share their achievements related to energy consumption and generation in their building with their friends on popular social networks. This would also be a disguised incentive for users to improve their awareness for energy conservation.

Finally, the current trend indicates that in the near future, more and more smart homes will be built and the technology inside them will grow very fast in order to create a more responsive and active environment able to respond to the users' needs [78]. Especially with the rising number of smart meters being produced and rolled out, real-time energy consumption information is being made available to an increasing number of household buildings. In order to make the best use of this valuable information and to gain increased benefit from it, it is anticipated that, in the future, smart home owners will have the option of joining together to form communities. They would then be able to share their in-house energy data with the communities for which they have obtained permissions and be informed about the energy use of like-minded residents in these communities in reward. Additionally, the

potential of offering gamification, i.e., the use of videogame elements and concepts from non-gaming contexts to improve user experience and engagement with an interface has started to be explored in recent years [78, 97]. Having the option of comparing personal energy usage with that of others may include a potential gamification element [78]. A community of smart homes can provide an ideal environment to facilitate such a gamification. "Support Online Community" is one of the use cases for smart homes that has been proposed in different articles (e.g. [107] and [1]). Consequently, it would be of advantage for the generic user interface if it were able to connect to different communities and display the various average energy usages of the communities to users.

However, it is noteworthy that the widespread deployment of smart meters has serious privacy implications since they inadvertently leak detailed information about household activities [85]. Therefore, while benefiting from the sharing of smart meter data with a utility company or a community, residents are also at the risk of exposing their privacy. To avoid this threat, smart meters should not transmit any sensitive data such as customer names or addresses, but to some extent, it will involve transmitting personal data through the use of a smart meter ID number, which can be associated with a recipient [108]. In addition, some privacy-preserving smart metering protocols could also be used to prevent from privacy violation of residents. For instance, the SMART-ER Protocol [63] achieves the goal of privacy protection by aggregating smart meter readings from a number of household buildings and then calculating masked readings for utility companies, whilst at the same time, ensuring providing utility companies with an accurate view on the current power consumption of their customers. Furthermore, recent work [108] proposes that certain initiatives should be undertaken, including (1) guidelines regulating access to data for customer services, (2) strong user control over information leaving the customer location and, (3) pro-

tocols that can process most of the data at customer locations, in order to assure customer privacy and data protection.

4.4 Environment Description

The generic user interface designed in this chapter is dedicated to serving smart homes in which a building operating system has been deployed to provide energy management services in order to achieve certain goals set by users, such as improving energy efficiency or reducing carbon emissions. A simplified model of the physical environment of the smart homes is illustrated in Figure 4.3. The building operating system in a smart home cannot work properly without the support of a smart meter which enables the real-time measurement of energy consumption in a building. The metering data can be sent back to utility companies or community service providers for the purpose of data analysis and statistics via the smart meter gateway by applying anonymization and encryption measures to ensure privacy.

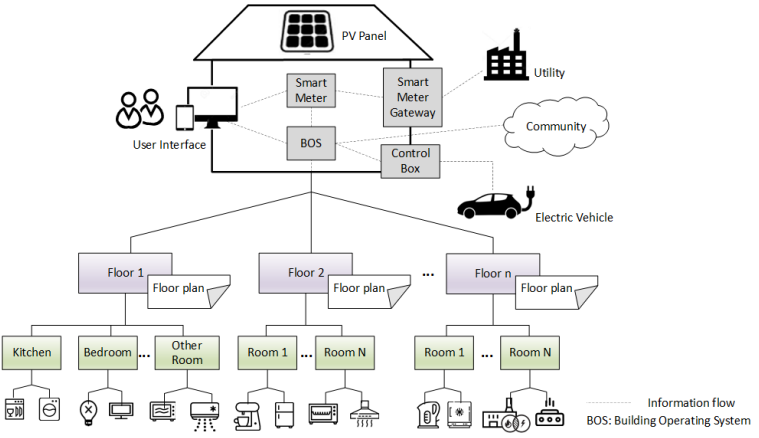


Figure 4.3: Environment of the smart home on which the generic user interface is oriented

The smart home on which the user interface designed in this chapter is oriented, is not limited to a single-storey household building. The user interface can also be applied to multi-storey buildings. For this reason, the smart home is modelled as a building which is composed of $n (n \geqslant 1)$ floors. If the value of n equals 1, it means the building is single-storey. If it is greater than 1, the building is multi-storey. Each floor consists of a series of locations (e.g. kitchen, living room, etc.) in which a variety of devices have been placed. A floor plan that shows the relationships between the different locations on a floor is assumed to be available for each floor. The generic user interface is to provide a logical and intuitive way for users to view their devices.

The devices in a smart home can be classified into different categories. From the perspective of schedule flexibility, home devices could be either reschedulable (e.g. washing machine, dishwasher, water heater, etc.) or non-reschedulable (e.g. oven, light, etc.). Users are allowed to set degrees of freedom for reschedulable devices (e.g. by specifying cut off time for expected programs) if their schedules can be shifted within certain range. The building operating system could decide on an optimal schedule for the devices to minimize the energy cost for users. However, since the generic user interface is to be user-centered, it will give users the right to overrule the optimized schedules and allow them to use the devices when they are needed. As for non-reschedulable devices, they have to start working as soon as they are needed by users. Therefore the generic user interface only needs to provide users with options to control these devices.

In addition to the conventional devices that consume electricity, many other devices which enable decentralized generation of renewable energy are becoming available in smart homes. For instance, photovoltaic (PV) panels can be installed on the roof of a smart home to convert the solar energy into electricity, and a Combined Heat and Power (CHP) unit can be used to produce heat and electricity simultaneously by being fueled with

natural gas. Furthermore, electric vehicles (EV) can also be integrated into the energy management of a smart home both in the form of consumer loads and electrical storage systems [87].

4.5 Roles

The home dweller, which represents all categories of persons who live in a home permanently, is thought of being the only home domain actor in the deliverable of smart buildings "scenario" definition [1] from the FINSENY project[2], and the authors later pointed out that distinctions can be made for more specialized roles/actors. In fact, the circumstances in a household building could be more complex than only having the role of home dwellers, who permanently live in the building. There are many conditions in which more than one role are needed to be dealt with. Some of them can be seen from the following exemplary scenarios.

Scenario 1: One family lives in a building and the father does not want his children to control devices at home via the user interface since they are too young to rationally use the devices.

Scenario 2: There are visitors coming into the home. The home owner is not comfortable if the visitors can see all the energy data in his building, so he wants to restrict the amount of information displayed on the user interface or limit their use of some devices via the user interface.

Scenario 3: The home owner owns the building and the appliances in it but he himself does not live in the building. Instead, he rents out all rooms to different tenants who do not want the home owner to track their energy use, in consideration of privacy protection.

Scenario 4: The home owner owns appliances in a building and also lives in the building. But he rents out rooms, which he does not need, to other

[2] http://www.fi-ppp-finseny.eu/

tenants. He will want to monitor his own energy use in the building through the user interface which is the same interface used by the other tenants. In addition to this, he also wants to be able to view the more global energy use of the building since he is the one to pay the energy bill.

There could be more complex scenarios, but even for the aforementioned simple scenarios, one role is not enough to meet the needs. In order to address the challenges resulting from these different scenarios, the users of the generic user interface are classified into three roles in this section: the administrator, the operator and the resident. Figure. 4.4 shows the relationship between these three roles.

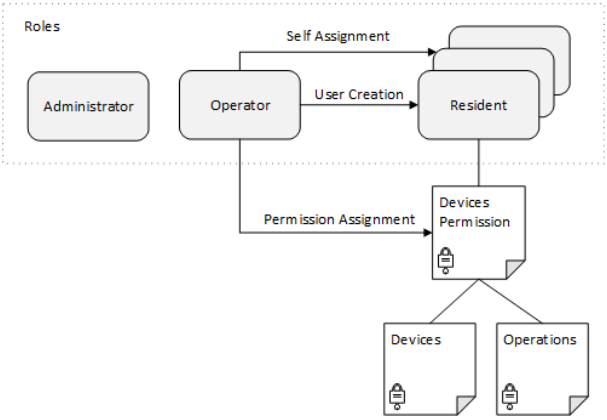


Figure 4.4: The three user roles of the generic user interface for building operating systems in smart homes

The administrator is in charge of the configuration of a building. His responsibilities in using the user interface include defining physical locations (e.g. floors and rooms) for the building, specifying relationships between the locations and arranging devices in their proper locations, in a virtual building within the user interface according to the actual situation in the building.

The resident is the role ascribed to users who are living in the building. Therefore a resident can use the user interface to control the home devices in the building to achieve various forms of home automation, and to view energy flows in the building as well as the energy data of devices of which he has been assigned corresponding operation permissions by the role of operator.

The operator is the role held by the one who needs to pay the energy bills for the building. The operator is responsible for managing the residents in the building, which means that the accounts for different resident roles in the user interface are created by the operator. He can also limit the residents' access to household devices by assigning residents with different permissions for the use of different devices. Each permission consists of two parts. One part refers to the devices that the residents have the right to access and the other part indicates the corresponding operations that residents are permitted to use in these devices. There are the following four kinds of operations which can be used to restrict the residents' access to devices:

- **View Device General Information**. This operation allows residents to view the general global information about the device. This information is usually static or not updated frequently, for example, the location of the device in a building, the time that the device was purchased, or some other factory information about the device.
- **View Device Channel Information**. The household devices in this thesis are considered to be made up of one or more so-called channels. A channel refers to an independent component of a device which can provide a certain function. A detailed description of channels can be found in the next section. This operation allows residents to view information about device channels which are usually dynamically changing, such as running states and power values, etc.

- **Control Device**. This operation allows residents to remotely control devices (e.g. switching on or off devices) via the generic user interface in order to realize different kinds of home automation.
- **Set Degree of Freedom**. The degree of freedom of household devices refers to the time interval within which the devices can be rescheduled [92]. The working schedules of some white goods in household buildings, such as washing machines or dishwashers, can be shifted either backward or forward within certain limits. With this operation, the residents can define these limits by specifying the time interval for running the devices on the generic user interface.

The scope of the permissions needed to perform the aforementioned operations increases progressively, and their relationship to each other can be seen in Figure 4.5. If a resident is allowed to view the channel information of a device, it implies that he can also view the general information of the device. If he is assigned the permission to control the device then he also obtains the right to view both general information and channel information of the device. Similarly, the permission for setting the degree of freedom for devices covers all the other operation permissions.

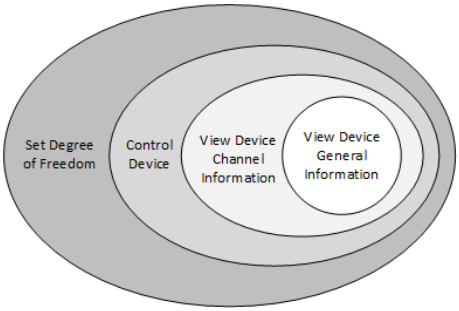


Figure 4.5: The permission hierarchy of the four operations for devices

If the operator also resides in the building, he can authorize himself to be a resident, which implies that he is given the possibility to assign himself permissions to access all the devices in the building by virtue of his privileges. To prevent the operator from misusing his privileges, the information concerning which users have the rights to access a device and what operations they are allowed to execute should be transparent to every resident for each device in the building. In this way, residents will be able to detect any invasive use or unauthorized monitoring of their devices.

The design idea of the roles in this section was inspired by Role-Based Access Control Models [98], whereby the author defined a family of four conceptual models for various dimensions of Role-Based Access Control (RBAC). The models in [98] have the common assumption that there is a single security officer, who is the only one authorized to manage the RBAC. However, the context of household buildings should be different. For reasons of personal privacy protection in household buildings, it is not reasonable to have one chief security officer. Only the people who reside in the building should have permissions to access the devices in the building. Even the operator, who is responsible for paying the energy bills in the building, is denied access the devices by the generic user interface, if he does not live in the building. The administrator and the operator are two independent roles, whose permissions are neither mutually exclusive nor inherited. Under certain circumstances, they could represent two different parties. It is certainly possible for one person to hold both roles at the same time. The simplest as well as the most extreme circumstance is that, one person owns the rights of the three roles. In this case, it means there is no role differentiation in a building, which is the normal situation in many households. Although the most common role in a household building is the resident, this does not cause much overhead to include the roles of administrator and operator in the design, and on the other hand, the addition of different roles is even necessary since they are needed in some special situations.

4.6 Data Models

As the central part of the architecture of the generic user interface for building operating systems (cf. Figure 4.2), flexible data models play a decisive role in making the user interface "generic". It is not challenging to design data models for specific devices in a certain building, as for the KIT EMP which is implemented specifically to cater for the ESHL. The problem of these data models is that they are not flexible to be extended to other buildings. Some of the current user interfaces for building operating systems (e.g. EF-Pi and OGEMA) are not even provided with data models. The data models for these user interfaces therefore need to be designed from scratch by UI developers. There are some building operating systems which provide generic data models for their user interface. However, the data models either cover only a few device types or support limited functionalities. For instance, in openHAB, the functionalities that are used by its user interfaces and automation logic are abstracted into so-called Items. The way of abstraction in openHAB is generic but since openHAB is designed for home automation solutions, its range of application is only limited to devices with simple functions, e.g. lights, players, roller shutters, etc. Complicated devices, like a washing machine, which can be assigned a degree of freedom, or a micro Combined Heat and Power unit (μCHP), which consists of different components, are not supported by openHAB. Another problem of openHAB is that the designer of its user interfaces need to know how the Items are defined in the openHAB instance in order to be able to design a user interface which again leads to the tight coupling between the user interface and the instance definition of the system.

In this section, a series of generic data models which are independent of any building operating system are designed in the scope of the generic user interface. These data models enable the generic user interface to not only serve various household buildings which are equipped with all kinds of

devices but also to support different kinds of home automation and facilitate a holistic overview for energy flows and devices in household buildings. To this end, a generic data structure, named HouseholdDevice, that models household devices for the generic user interface is designed. Its UML class diagram is shown in Figure 4.6.

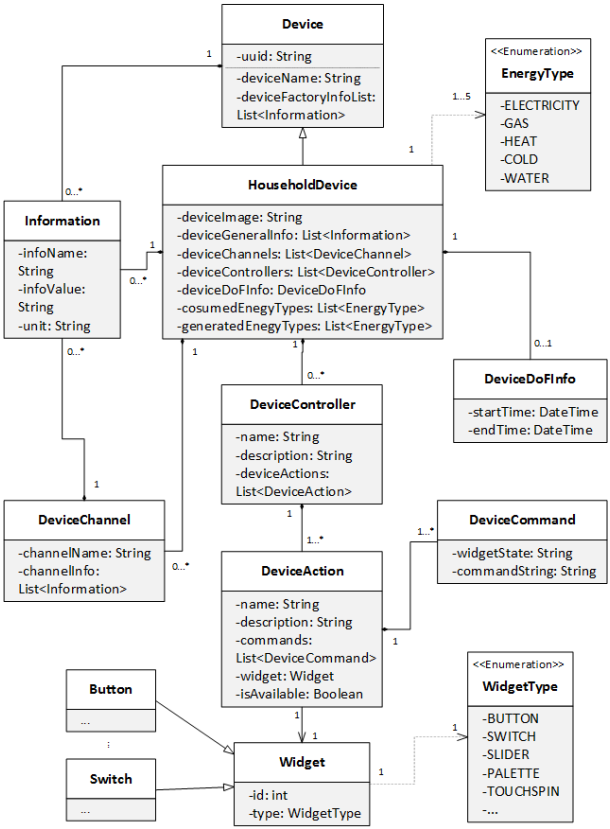


Figure 4.6: The UML class diagram of the household device model

The class HouseholdDevice is derived from a parent class, named Device, which consists of the basic information about a device. The attributes of the

Device class include a 'uuid', which is an identifier that uniquely identifies this particular device in a building, a device name which can be changed later by users, and a list of the device factory information. In the Device class, the factory information is not specific since this varies from device to device and is full of unknowns. Instead, for the generic user interface, a piece of information is abstracted by a special Information class which consists of a name of the information, a value corresponding to the name, and a unit for that value which can be empty when it is not needed for the information. By using this abstract representation, the Device class can be used to integrate any factory information about a device. The generic user interface does not need to know what exact information is contained in the list. Instead, it simply needs to iterate the list and display each piece of information in the list on its web page. Table 4.1 shows a washing machine instance of the Device class.

Table 4.1: An exemplified Device instance of a washing machine

Device Id	1609555631		
Device Name	Washing Machine		
A List of Factory Information	**Information Name**	**Information Value**	**Unit**
	Producer	Robert Bosch GmbH	
	Brand	Bosch	
	Model	WAK28227	
	Energy Efficiency Class	A +++	
	...	...	...

In addition to the attributes inherited from its parent class, the class HouseholdDevice has an attribute 'deviceImage', which records the address of the device image in the web server. To be able to display across-the-board information for all kinds of devices including some large complicated ones, the household devices are considered to be made up of one or more channels. A channel in this chapter refers to a functional component of a device.

Most household devices, such as lights or the television, provide only a single function and therefore only have one channel, where the current status, power value or some other information about the device can be stored. Many devices are more complex and are made up of different components which can provide their own independent functions. For these devices, more than one channel is needed to represent the different components and to store various kinds of information about each. For instance, a dual-temp refrigerator can be considered to contain two channels, which correspond to its refrigerator and freezer compartments, respectively. Another example is the μCHP, where a boiler and a small power plant are combined in a single durable device, so that the boiler and the power engine are abstracted into two channels of the μCHP. Sometimes the μCHP might be extended with an electrically driven heating coil as an alternative actuator to produce heat. In this case, a third channel is needed to represent the heating coil.

In the HouseholdDevice model, the attribute 'deviceChannels' is the collection of all channels in a device. A channel is modelled as a DeviceChannel class which consists of two attributes: 'channelName' which is the name of the channel and, 'channelInfo', which is a list of information about the channel. The information about a channel is also represented by a list of Information objects. Table 4.2 shows an example of three channels of a μCHP. The information about the channels is provided by the building operating system and will be further displayed on the generic user interface.

In addition to the information about device channels, there is some global general information about the devices. This general information does not merely apply to a single channel but to the device as a whole, such as the location of the device, or the list of residents who are allowed to access the device, etc. An attribute named 'deviceGeneralInfo', which also consists of a list of Information objects, is used to store this general information of a device.

Table 4.2: Example of device channels of a μCHP

Channel Name	Channel Information		
	Information Name	**Information Value**	**Unit**
Power engine	State	Running	
	Power	-7400	W
Heating Cartridge	State	Running	
	Power	2000	W
Water Boiler	Top Temperature	77	°C
	Middle Temperature	41	°C
	Bottom Temperature	39	°C

With decentralized generation increasingly becoming an option in household buildings, the role of the smart home will gradually make a transition from being a conventional consumer to a future prosumer. At the same time, the home devices will also become increasingly diverse. In order to facilitate the classification of devices, based on energy types which they support, on the generic user interface, the HouseholdDevice model provides two attributes, i.e., 'consumedEnergyTypes' and 'generatedEnergyTypes'. These two attributes represent a list of energy types that the corresponding devices consume, and a list of energy types that they produce, respectively. The possible energy types include electricity, gas, heat, cold and water.

In addition to displaying information about the device on the generic user interface, the HouseholdDevice model provides possibilities for users to interact with devices. In order to support the setting of degrees of freedom for devices, it is equipped with an attribute named 'deviceDoFInfo', which consists of an 'allowed start time' and a 'required end time' for the running program of devices. The values will be specified by users on the user interface.

Except for sensors, most of the household devices provide one or more functions that can be controlled by residents. Each of the functions is consi-

dered to be a controller which is abstracted into a DeviceController model. The HouseholdDevice model contains an attribute named 'deviceControllers' which represents a list of DeviceController instances. For example, some multifunctional air conditioners can regulate and control the temperature, humidity and cleanliness of the air, so that, in this case, each of the functions provided by the air conditioners corresponds to a controller which can be modelled by a DeviceController class. The DeviceController contains attributes including a name of the controller, a description of the controller and a list of DeviceAction instances. The DeviceAction is an abstract representation of a function unit of a device controller. In addition to a name and a description, attributes of the DeviceAction class include a Widget instance and a list of DeviceCommand instances relating to the Widget. The Widget is a base class for user interface elements. Specific interface elements such as buttons, switches or sliders, etc. can extend the class by adding their own attributes. The DeviceCommand model includes the attributes of a widgetState and a commandString, which records which command (covered by the attribute 'commandString') should be sent to the building operating system and in which state of the Widget (covered by the attribute 'widgetState'). When the state of the Widget of a device is changed by residents, the generic user interface will find the command string corresponding to the widget state, and send it to the building operating system so as to realize the control of the device. In addition, another attribute, name 'isAvailable', is also attached to the DeviceAction class. This attribute is used to indicate whether the device action is available on the user interface. For instance, users should not turn on the dishwasher via the user interface when it is already in a state of running. In this case, the action of turning on the dishwasher should be set to 'not available' (*isAvailable* = *false*), so that the user interface will disable this option for users.

According to the Electric Vehicle Outlook 2017[3] from the International Energy Agency (IEA), the global electric car stock surpassed two million units in 2016 after crossing the one million vehicle threshold in 2015, and the number has been growing continuously. These electric vehicles can be connected to household buildings via charging stations when they are not in use. While the charging process of the car's battery can be controlled by the building operating system to prevent grid overload, the battery can also be used as storage for electrical energy [46]. Since electric vehicles are different from the conventional household devices, the HouseholdDevice model cannot cover the scenario of electric vehicles. In this case, a special ElectricVehicle class (cf. Figure 4.7) is designed for them.

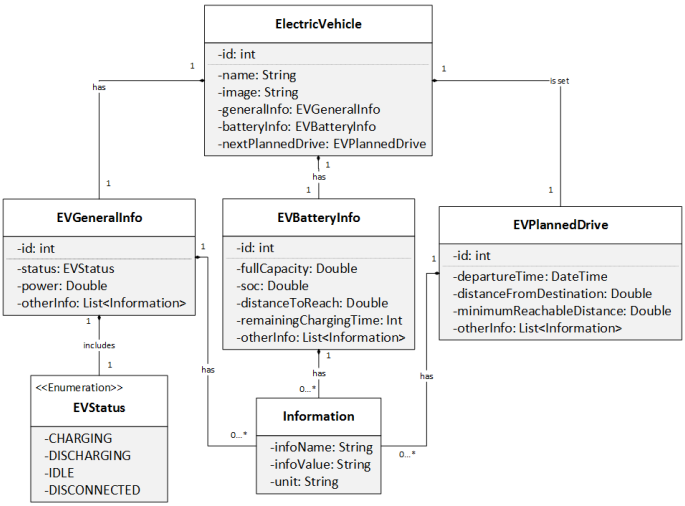


Figure 4.7: The UML class diagram of the electric vehicle model

Besides an identifier (id), a name and an image of an electric vehicle, the ElectricVehicle class includes a 'generalInfo' attribute, which is an instance of the EVGeneralInfo class, in order to cover the basic generality

[3] https://www.iea.org/publications/freepublications/publication/GlobalEVOutlook2017.pdf

information about an electric vehicle, such as the current status and power value. The potential status of an electric vehicle could be either connected or disconnected to a charging station. In case that the electric vehicle is connected to the charging station, it may be in one of the following three states: charging, discharging and idle. In addition to the status and power, other general information about an electric vehicle can be included in the 'otherInfo' attribute. The 'otherInfo' attribute is a list of customized Information instances, which have been explained in the HouseholdDevice model. The detailed information about the battery of an electric vehicle is covered by the 'batteryInfo' attribute, which is an instance of the EVBatteryInfo class. Attributes of the EVBatteryInfo class include the full capacity of the battery, the current state of charge (SOC), the distance that the car can drive with the current battery capacity, the remaining charging time and a list of other information.

Although an electric vehicle can be used as a flexible mobile energy storage device in a household building, such a potential of the electric vehicle can only be exploited by the building operating system on the premise of fulfilling the needs of users. Concerning this, the ElectricVehicle model provides another attribute named 'nextPlannedDrive' to record the parameters configured by users for their next drive. These parameters include a departure time for the next drive, the distance from the target location, the minimum range that the car needs to reach, etc. All the requirements specified by users indicate a degree of freedom of the electric vehicle.

As discussed in Section 4.5, the users of the generic user interface designed in this chapter are classified into three roles, namely, the administrator, the operator and the resident. A so called User class (cf. Figure 4.8) models the users of the generic user interface in a building. Since the generic user interface aims to implement role-based access control, a user of the generic user interface is allowed to cover more than one role. Therefore, in addition to the basic attributes such as user name, password and some personal data

about the user, the User class also has an attribute to indicate a list of roles that the user owns. Residents live in a building and use the household devices on a daily basis, however they are only allowed to use the devices that the operator has assigned to them in a specified way. Therefore the class, Resident, extends the User class by adding a list of permissions for the resident to operate devices in a building. The permission is modelled by a Permission class, which contains a list of household devices as well as a list of operations allowed by the devices.

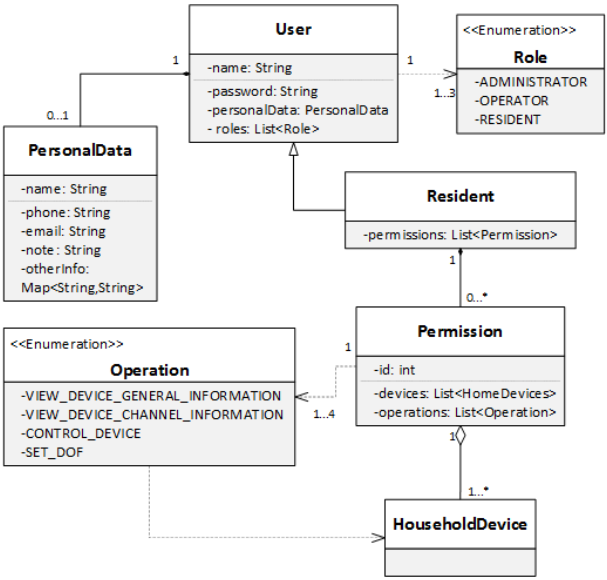


Figure 4.8: The UML class diagram of the user model

As described in Section 4.4, a household building in this thesis is considered to have one or more floors. A single apartment is regarded as a building having one floor which is the apartment itself. Figure 4.9 shows the UML class diagram of the floor model. The floor model represents the

global configuration of a building which therefore will be managed by the administrator via the generic user interface.

A Floor class is made up of the following attributes: an id, which uniquely identifies this floor in a building, a name of the floor, which is defined by the administrator, a list of locations (e.g. rooms or hallway) and a floor plan. Locations in a floor are modelled by a Location class which has attributes of an id, a name, which is defined by the administrator, and a list of household devices, whose data model is shown in Figure 4.6.

In order to give users an intuitive and overall overview of the devices in their building, one of the design concepts of the generic user interface in this thesis is to display the users' household devices on the floor plan of their building. This concept was inspired by PGM2, the user interface of FHEM, and the EMP, the user interface of the OSH (cf. Section 3.1). A FloorPlan class (cf. Figure 4.9) is designed for this purpose. The floor plans of a building on the generic user interface are represented in the form of images. The FloorPlan class defines a group of basic attributes pertaining to a floor plan image, including address, width, height and scale of the floor plan image. The reason for having a scale attribute is to enable users to scale up or scale down the original image of the floor plan on the user interface according to their needs.

Another attribute of the FloorPlan class is a list of devices that are placed on the floor plan by the administrator. The images of the household devices in a building can be placed in corresponding positions on its floor plan according to the actual location of the devices in the building. The devices on the floor plan are abstracted into a class named DeviceOnFloorPlan which contains attributes of the device images that are on the floor plan, such as their coordinates, width and height. In order to visualize different states of devices on a floor plan and provide anchors for resizing the device images, every device on a floor plan is designed to be surrounded by four small circles which are attached to the four corners of the device image on the floor

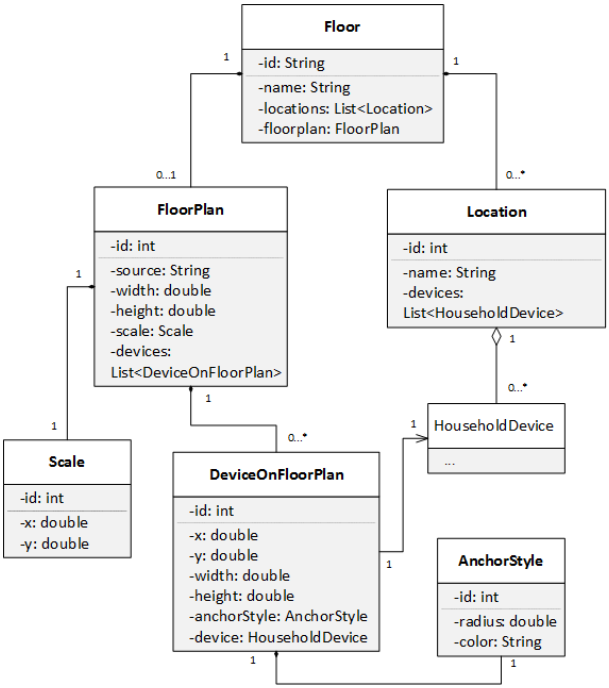


Figure 4.9: The UML class diagram of the floor model

plan. The style of the circles is specified in the class AnchorStyle which includes a radius attribute and a color attribute to specify the size and color of the circles. Different colors of the circles are used to indicate different states of a device. Furthermore, each device on a floor plan is associated with a particular household device in the building so that detailed information about the device and the executable controls over the device is achievable. For this purpose, the DeviceOnFloorPlan class has a 'device' attribute which is a reference to a HouseholdDevice instance. Because of this, users are able to not only view devices in their building but can also interact with them on the basis of a floor plan.

Figure 4.10 shows data models relating to household buildings. Firstly, the Building class contains some static information pertaining to the entire building or information that is not updated frequently. Attributes of the Building class include a list of global information (e.g. floor area, year built, etc.) related to a building, a list of Floor instances and a list of stakeholders of a building which is a collection of the users of the three roles (i.e., the administrator, the operator and the resident) of the generic user interface in a building. In addition to static data, the BuildingRealtimeData class is used to model the real-time changing data in a building. Its attributes include a timestamp which identifies when the current data was generated, voltage at a building, frequency at a building, power consumption of the whole building, power generation of the whole building and a list of other dynamically changing information about a building. The values of the attributes will be provided by a building operating system, and the generic user interface will be responsible for displaying them for users.

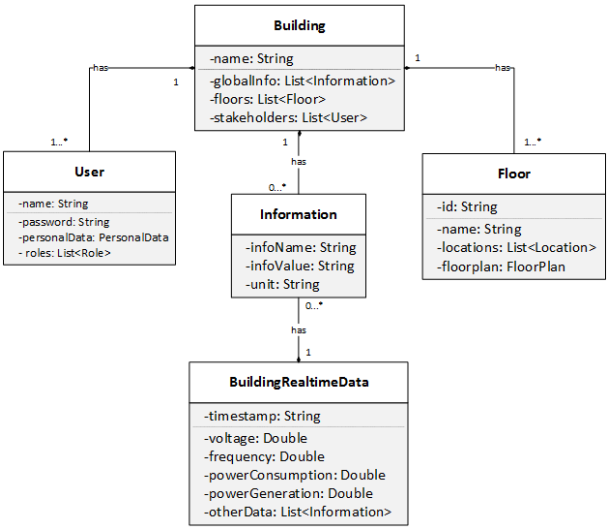


Figure 4.10: The UML class diagram of building related models

As one of the basic use cases in the context of smart home, simple home automation, such as changing the state of a specific device (e.g. switching a light on/off) can be implemented based on the HouseholdDevice model (cf. Figure 4.6). In addition, the generic user interface designed in this chapter allows users in a smart home to realize some more advanced home automation. One of them is to allow residents to organize a number of devices into a group in order to facilitate residents to view the status of these devices or to control them in a uniform way. The devices which are used frequently by residents or devices of the same type may be classified as a group. Figure 4.11 shows the class diagram of the model of the device group.

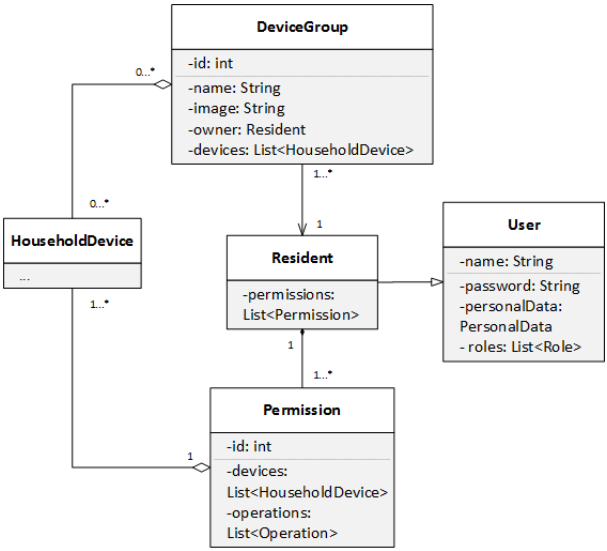


Figure 4.11: The UML class diagram of the model of the device group

A device group is owned by the resident who created the group. Residents can specify a group name and an image for a device group. That is why the DeviceGroup model includes attributes of 'a name', 'an image' and 'an owner'. Besides this, it also includes a list of household devices which

are added to the group by the owner of the group. Since the residents' access to the household devices in a building is restricted by the operator via certain permissions, residents can only make groups for those devices that they have permissions to access. After having defined a device group, on the one hand, residents are able to view the states of the devices in the group with one glance. On the other hand, devices in the group can be controlled by residents by using unified group operations, provided that the devices belong to the same type. This is due to the fact, that controllers of the same type of devices are represented by the same widgets on the generic user interface. Due to the flexibility of the HouseholdDevice model, the widgets therefore can be extracted from the corresponding HouseholdDevice instances of these devices and can then be included in group operations for residents to be able to control all the devices in the group simultaneously. For instance, all the lights in a building can be organized into a group. The common widgets relating to the lights could be a switch component or two buttons which can be used to switch a light on and off. By having the common widgets as group operators, residents are able to turn on or off all the lights in the group simultaneously instead of having to control them individually.

In addition to device groups, the generic user interface enables residents to create various scenes in their building according to their needs. In the general context of home automation, a scene is a defined set of states of one or more home devices, and an example of such a scene could be a night scene, which turns on all the indoor lights [24]. After being created, scenes can be triggered by users whenever they need. The class diagram of the scenes is shown in Figure 4.12. A scene is owned by the resident who created it on the generic user interface. The resident can add devices, that they have permissions to control, to the scene. Therefore, the Scene model consists of the following attributes: an 'id' which uniquely identifies this scene, a 'name' of the scene which is given by the owner of the scene, a 'resident' who created the scene and owns it, and a 'list of devices' that

have been added to the scene by the resident. The residents in a household building are abstracted into a Resident class which extends the User class.

The device that has been added to the scene is abstracted by the class DeviceInScene which is associated with a HouseholdDevice instance. As shown in Figure 4.6, if a household device is controllable, its data model HouseholdDevice will contain one or more DeviceController members. The DeviceController model further has one or more DeviceAction members. Each of them contains a Widget instance and a list of DeviceCommand instances which store different widget states and their corresponding command strings that need to be sent to the building operating system in order to control the devices to physically reach the corresponding states. After adding some devices to a scene, on the generic user interface, the resident who owns the scene can specify the target states for the devices. These target states, together with their command strings, will be stored in a list of TargetStateAndCmd instances, which is another attribute of the DeviceInScene class. The TargetStateAndCmd class contains a 'controllerName', attribute which identifies a DeviceController instance of the device in a scene, an 'actionName', which identifies a DeviceAction instance in that DeviceController and a 'commandID' attribute, which identifies a DeviceCommand instance in the DeviceAction. Through these three identifiers in the HouseholdDevice model, the generic user interface can easily find the device's target state that was set by the resident and the command string corresponding to the target state, which needs to be sent to the building operating system, by accessing the HouseholdDevice model.

When a resident triggers a scene that he has created, the generic user interface will traverse a list of TargetStateAndCmd instances for every device that has been added to the scene and get the command string for the target state of the device. It will send this to the underlying building operating system which will translate it into specific instructions and further send these to the device in a building so as to control the device to reach the desired

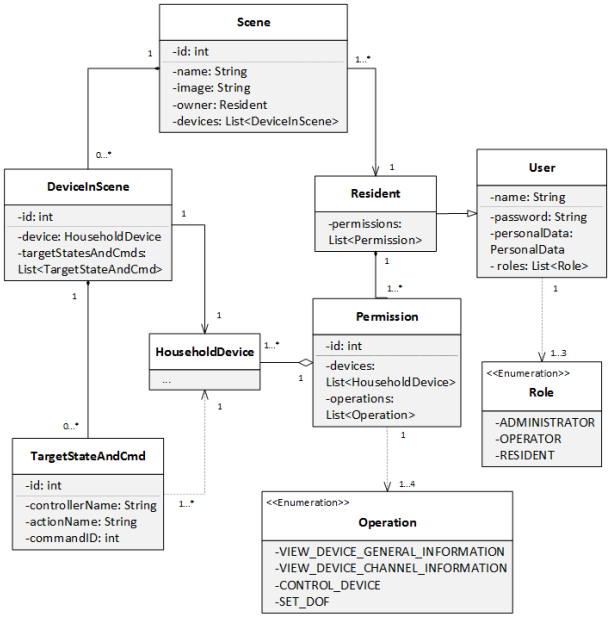


Figure 4.12: The UML class diagram of the scene model

target state. The detailed process of adding and triggering a scene can be found in Figure 4.13.

The generic user interface in this chapter is designed to support a number of advanced home automations aimed at improving the residents' comfort in their building. In addition to supporting the creation of scenes, it also allows residents to create automation events on the user interface, with the aid of a normal calendar. This chapter abstracts the automation events on a calendar into a CalendarEvent model, whose UML class diagram is shown in Figure 4.14. The CalendarEvent model is composed of the following attributes: an 'id', which uniquely identifies the event, a 'title' of the event, a 'resident', who created the event, the 'date and time' of the beginning and end of the event, the 'location' settings for the beginning and end of the event, a 'list

of device settings' for the beginning and end of the event, and the 'repeat mode' of the calendar event.

The settings concerning a location of a calendar event are modelled as a LocationSetting class, which consists of the following attributes: an 'id', which uniquely identifies the location setting, a 'location' in a building, a 'temperature value' of the location, a 'humidity value' of the location, a 'Boolean value' (i.e., 'isTempRequired'), which indicates if the temperature value of the location is required for the event and a 'Boolean value' (i.e.,

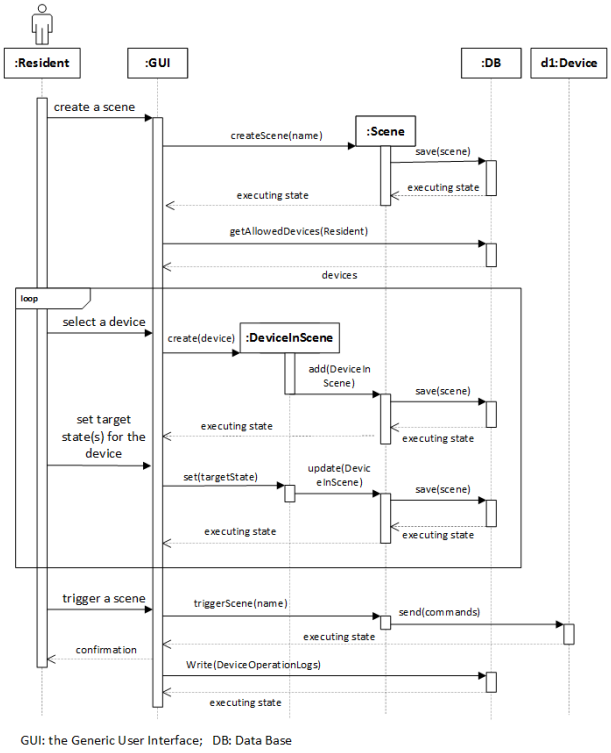


Figure 4.13: The sequence diagram of creating and triggering a scene

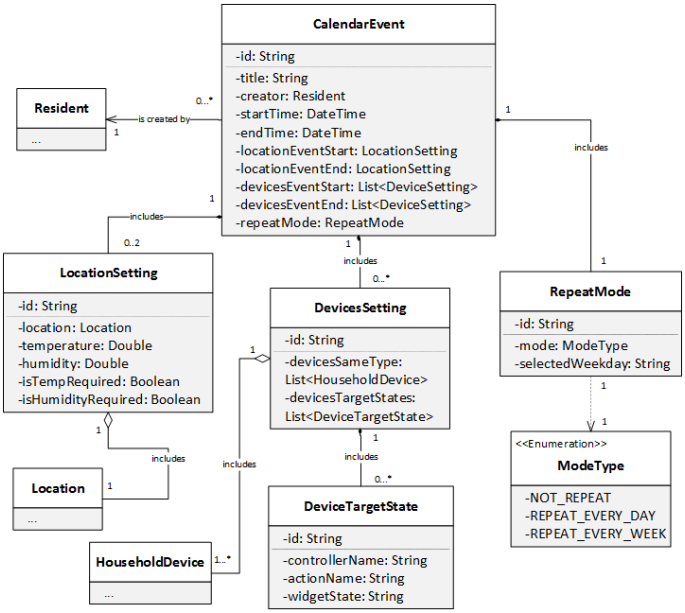


Figure 4.14: The UML class diagram of the calendar event model

'isHumidityRequired'), which indicates if the humidity value of the location is required for the event. If the value of the 'isTempRequired' attribute or the 'isHumidityRequired' attribute is set to false, the building operating system will not consider optimizing the temperature or the humidity in the location.

In addition to setting a location for the beginning and end of the event, residents are also allowed to configure the state of devices for the event. The settings related to the devices of a calendar event are modelled as a DevicesSetting class. In general, the same type of devices in a location (e.g. all the lights of a room) need the same target state for a certain event. In this case, these devices can be configured as a whole. Therefore, a list of household devices is one of the attributes of the DevicesSetting class. Since household devices could provide different controllers to interact with users,

the DevicesSetting class includes an attribute of a list of 'deviceTargetState' instances to cover different target states of a set of devices of the same type.

The last attribute of the CalendarEvent is the repeatMode which indicates in which way the calendar event needs to be repeated. For each calendar event, it can be set to not repeat, which means the event will only occur once. It can also be set to repeat every day or every week. If the mode of repeating every day is chosen, the event will be triggered every day at a fixed time within the given period of dates. If the mode of repeating every week is chosen, the resident has to specify a weekday for the event. In so doing, the calendar event will be triggered at a fixed time on the same day of every week within the given period of dates. By triggering a calendar event, it means the devices that have been added to the event will be automatically set to the specified target states.

The settings that are specified for a calendar event are supposed to be taken care of by the generic user interface as well as the building operating system. The generic user interface is supposed to trigger the target states of the devices in the event, by sending the command strings corresponding to the target states to the building operating system. The settings (i.e., the temperature and the humidity) of a location for the event are supposed to be taken care of by the building operating system, since these settings are equivalent to the degrees of freedom of an air-conditioner in the location. The working process of the air-conditioner is schedulable in order to reach the setting goals of the location for the event. This provides possibilities for the building operating system to realize load shifting. In the generic user interface, two threads are needed in order to implement the functions of integrating events into a calendar. The first thread is responsible for accepting the residents' instructions to manage (including create, view, update and delete) events on the calendar. Another thread ensures that the events that have been created on the calendar are activated on time. Their sequence diagram can be found in Figure 4.15 and Figure 4.16, respectively.

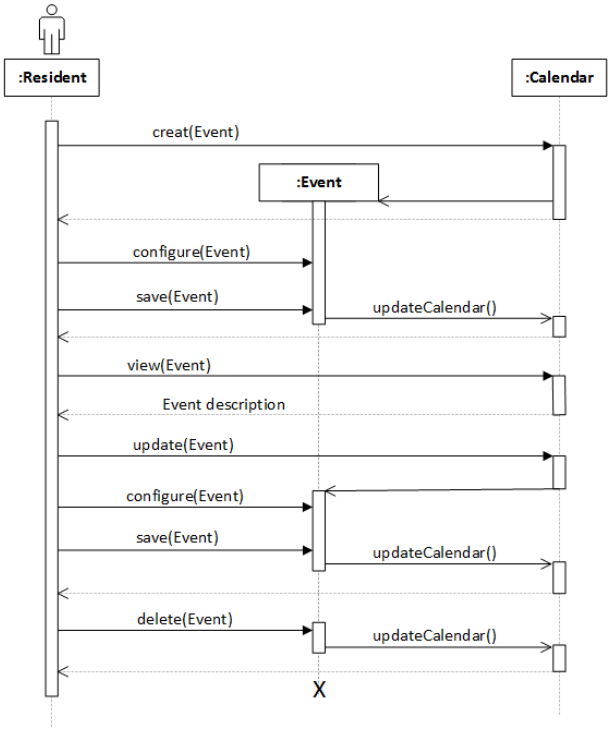


Figure 4.15: The sequence diagram for creating and managing an event via the calendar of the generic user interface

As described in Section 4.3, the owner of a smart home in the future may choose to join one or more communities so that they can compare the energy usage of their building with that of their counterparts in a community. To achieve this, the communities are abstracted into a Community class (cf. Figure 4.17), which will be used by the generic user interface to not only display information about the communities for users but also provide options for users to configure connections to the communities.

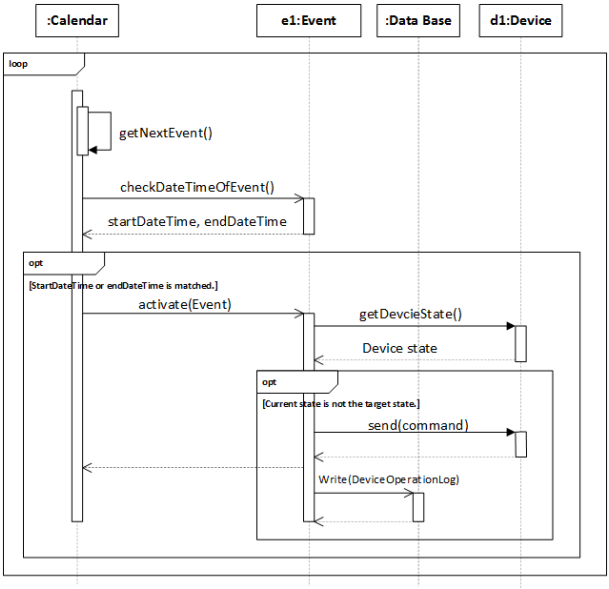


Figure 4.16: The sequence diagram for realizing advanced home automation via the calendar in the generic user interface

The attributes of the Community model include a 'name' of the community which also uniquely identifies the community and a logo of the community. Additionally, a 'connectionStatus' attribute indicates the connection status between the building and the community which could be one of the following: joined, connected, disconnected and unjoined. Depending on the current connection status, proper actions can be made to change the connection status. The optional actions include joining a community, connecting to a community, disconnecting from a community and quitting from a community. For instance, if the building has not joined a community, the home owner can choose to join the community. After that, the building is still disconnected from the community by default which means the energy data in the building will not be sent to the community for statistics. As a

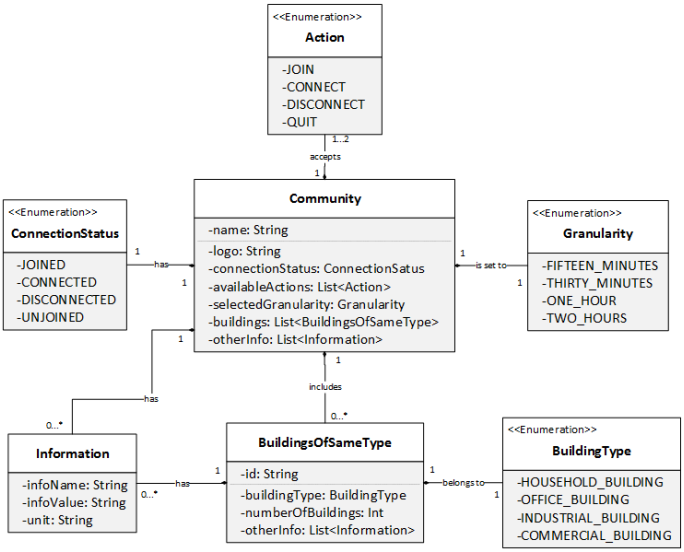


Figure 4.17: The UML class diagram of the community model

result, the home owner does not have access to the average energy data from that community. In order to connect to the community, a manual connection needs to be made by the home owner via the generic user interface. Under special circumstances, he can also cut off the connection or even quit from the community. In the Community class, the 'availableActions' attribute covers the available actions that the home owner is allowed to make based on the current connection status. Furthermore, the Community model provides a 'selectedGranularity' attribute which indicates the level of the data granularity (i.e., time intervals) for uploading the energy data of the building to a community. The value of the attribute is configured by the home owner to ensure privacy protection.

A community is usually a collection of different kinds of buildings. In order to provide home owners with the general information about the buildings in a community, the Community model includes an attribute named

'buildings', which is the combination of different types of buildings. The information about the same type of buildings is abstracted into a BuildngsOfSameType class. It includes an attribute to indicate a building type, for example, a household building, an office building, an industrial building or a commercial building, and another attribute to show the number of buildings of this type which exist in the community. Other information about buildings of a certain type and about the community is covered by a list of customized Information instances.

One of the most important features that the generic user interface needs to have is to provide residents with a holistic and intuitive overview of the current energy use in their building. Current user interfaces for building operating systems basically organize the household devices on the basis of their locations in the building. It is clear that organizing devices in this way makes it logical and intuitive for residents to be able to find devices. However, it is not supposed to be an intuitive way when it comes to residents desiring to get a full picture of the energy use in their building since the different kinds of devices that are running are scattered in many different places. The HouseholdDevice model (cf. Figure 4.6), which contains consumed energy types and generated energy types as its attributes, enables the generic user interface to organize devices in a different way, so as to help residents to have a clearer understanding of the energy use in their building. The in-house energy overview provided by the generic user interface is classified into multiple levels which are organized as a tree structure (cf. Figure 4.18).

The first level of the energy overview tree is about different energy types, which include electricity, heat, cold, gas and water. Under each energy type in Figure 4.18, the second level, which consists of various running modes of the different devices for this energy type, can be seen. So, for each type of energy, devices in a building could be consuming the energy, generating the energy, or be in a state of storage. For instance, for the energy type of electri-

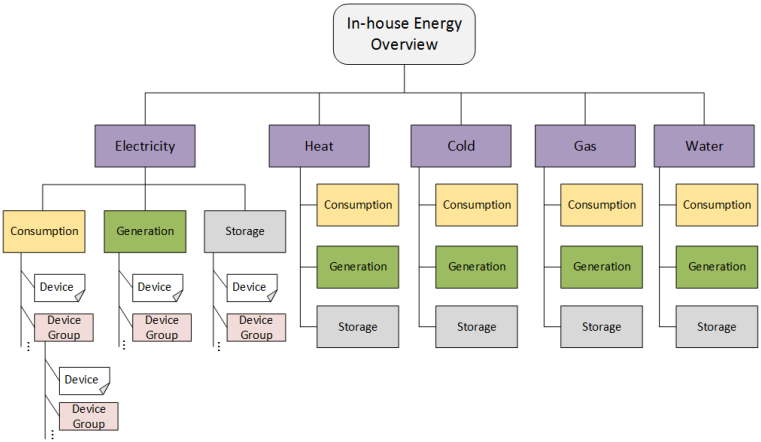


Figure 4.18: The structure of the energy overview provided by the generic user interface

city, household devices could be classified into types of power consumption (e.g. lights, TVs, etc.), power generation (e.g. photovoltaic panels) and storage (e.g. batteries) according to devices' real-time running state. The third level refers to either devices or device groups which are a group of devices defined by residents according to their preferences. The members of device groups could be devices or nested device groups. For example, a light group could be defined to contain all the lights in the building. Within the light group, a nested group could be further defined to contain lights in a certain room. It is worth mentioning that some multi-modal household devices, which can deal with different kinds of energy forms, will be grouped into different energy types at the same time whenever they are in operation. This means that a device may appear in different places of an energy overview tree at the same time. For instance, a μCHP, as a co-generation plant, will simultaneously appear in the following locations of the energy overview tree when it is running: Electricity/Generation, Heat/Generation and Gas/Consumption. If a μCHP is extended with an electrically driven hea-

ting cartridge, it will also appear under the path of Electricity/Consumption when the heating cartridge is working. The data structure used by the generic user interface to render the energy overview tree is represented in the form of JavaScript Object Notation (JSON). A JSON snippet can be found in Figure 4.19 which shows an exemplified representation of part of the energy overview tree showing the devices that are consuming electricity in a building.

JSON is made up of a collection of key/value pairs. The generic user interface will render the energy overview tree according to the JSON representation provided by a building operating system. The type of 'energy group' in the JSON representation indicates the root of an overview tree for a certain type of energy. The type of 'running mode' refers to a subtree in the second level. The type of 'device group' refers to a subtree in the third level. The values corresponding to the key 'children' in different levels represent children of the corresponding subtree. The type of 'device' refers to the leaf node of a tree, which has no more children. A device can be directly a child of subtrees about a 'running mode' (e.g. the oven and the dish washer in the JSON snippet above) or a child of subtrees about a 'device group' (e.g. the lights in the JSON snippet above). If a device changes its state from idle to running, a new record about the device will be appended as a child to the JSON file. Similarly, if a device stops running, its related record will also be removed from the JSON file. The overall energy consumption/generation/storage for each tree/subtree in a building can be calculated by collecting corresponding energy values from the devices in that tree/subtree. By doing so, residents can have a holistic overview of the energy usage in their building from the energy overview tree. In addition to this, residents have access to detailed information about a specific device. This can be achieved with the aid of the identifier of devices which corresponds to the value of the key 'id' of each device record in the JSON file.

```
{
    "name": "Electricity",
    "icon": "electricity.png",
    "type": "energy group",
    "children": {
    {
        "name": "Consumption",
        "icon": "powerConsumption.png",
        "type": "running mode",
        "children": {
                {
                    "name": "Lights in my home",
                    "icon": "lights.png",
                    "type": "device group",
                    "children": {
                                {id": "1609551312", "name": "light_kitchen", "icon":
"light.png", "type": "device"} ,
                                {id": "1609554862", "name": "light_livingRoom",
"icon": "light.png", "type": "device"} ,
                                {id": "1609555628", "name": "light_bedroom",
"icon": "light.png", "type": "device"}
                            }
                } ,
                { "id": "1609551312", "name": "dish washer", "icon": "dishWas-
her.png", "type": "device" } ,
                { "id": "1609557469", "name": "oven", "icon": "oven.png", "type":
"device" }
            }
    } ,
    ...
}
```

Figure 4.19: An exemplified JSON representation of part of the energy overview tree which shows devices that are consuming electricity

By fragmenting the whole 'level' of the energy usage in a building into different levels, it is on the one hand straightforward for residents to get a clear picture of the global energy flows in their building. On the other hand,

the tree-structured organization provides a flexible and scalable way of displaying the devices in a building. Adding or removing a device is equivalent to adding or deleting a leaf node from the energy overview tree. In addition, the generic user interface renders the energy overview tree by parsing the JSON representation according to the key/value pairs from a building operating system, which further ensures generality of the user interface.

4.7 Functional Components

As shown in Figure 4.1, the content is an essential part of a user interface design. Based on the requirements (cf. Section 4.1) of the generic user interface for building operating systems, this section introduces the detailed functional components (cf. Figure 4.20) of the generic user interface.

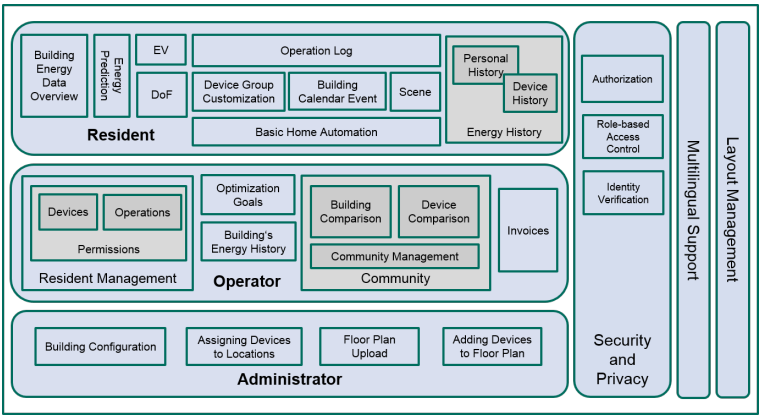


Figure 4.20: The functional components of the generic user interface for building operating systems

As was described in the previous section, three roles are created for various responsibilities in the household building: administrator, operator and resident. The administrator is responsible for configuring the building by

adding floors to the building and adding locations to the floors, assigning devices to appropriate locations, uploading floor plans for the building and adding devices to corresponding floor plans. As soon as the configuration is complete, the generic user interface is ready to be accessed by the operator and the resident.

The operator is responsible for managing residents and paying energy bills in the building. He can assign different permissions to different residents, which means that the residents can be restricted to perform only those operations on certain devices for which they have received authorization. Since the operator needs to pay the energy bills, he can, besides checking the invoices sent from the utility company on the generic user interface, also set optimization goals for the building, e.g. minimizing the energy costs, energy consumption and CO_2 emissions or maximizing energy consumption from renewables, etc. It is possible for the operator to be able to set multiple (even conflicting) optimization goals by having him specify weightings for different goals. The building operating system is supposed to balance these goals and define a reasonably good trade-off between the competing goals. In order to be able to get a better impression of the energy use in the building, the operator can check the building's energy consumption and generation history by using electricity prices and load limits as reference. In addition to this, the operator can choose to join "communities" to exchange information, or involve in gamification and statistic calculation. In return for sharing some of the energy data in the building with communities, he can compare the energy use as well as the utilization of devices in his building with the corresponding statistical average values of the other buildings in the community. The per capita power consumption or generation in a building could be used for comparing the energy use in a community. The devices in a building can be compared according to different aspects, such as average power use per person, average cost per person, average power use per usage, and average cost per usage, etc. After a comparison such as

this, the operator will be able to not only become aware of the state of the overall energy use in his building but also notice whether a particular device in his building is being utilized appropriately by the residents or whether devices in his building are energy efficient when compared to those of other buildings in a community.

The residents, who are living in the building, can view the real time energy data as well as the predicted energy data of the building along with various external signals, including energy tariffs and load limits. As for the historical energy information, residents can view both their personal historical energy use or the historical data of a specific device. Besides this, they can also configure the next drive for the electric vehicle and set degrees of freedom for devices. In addition to the function of basic home automation, the generic user interface enables residents to implement advanced home automation, which makes possible the execution of multiple actions at specific times, locations, or events, thus allowing them to manage their building in a smart way.

Firstly, it allows residents to customize their own device groups. For instance, they can group all the lights in the building into one light group, so that they will be able to control all of the lights by sending commands to this group. In addition to this, the residents can create different scenes according to their needs. They can add devices to the scene and specify target states for the devices via the generic user interface. After configuration, a scene can be triggered any time by the resident who created it. Furthermore, the generic user interface provides a calendar service which plays an important role in the energy optimization of the building. The residents' schedule information, marked on the calendar, can be used as auxiliary information for the optimization process of the building operating system. From the residents' point of view, the calendar service enables them to realize advanced home automation by allowing them to add calendar events. For instance, a resident can add an event on the calendar to roll up the blinds in the building

in the morning and roll them down in the evening on a specific day or to repeat this every day. Another example would be a party, which is a more complicated event. For such an event, the resident can, on the one hand, set parameters (e.g. temperature or humidity) on the calendar for the location where the event is to take place. On the other hand, he can select the devices needed for the event and set target states for these devices. The generic user interface, together with the building operating system, are supposed to take care of the event on the calendar and make sure the settings will be met when the event begins. Finally, all operations on the devices in a building will be recorded in an operation log, which is allowed to be checked by residents via the user interface.

In terms of the system as a whole, the generic user interface provides authorization, role-based access control and identity verification in order to ensure security and privacy. Moreover it provides multilingual support which is helpful to users who prefer different native languages. And, last but not least, the generic user interface provides layout management to ensure that the user interface will look good on different-sized screens so as to improve the user experience.

4.8 Conclusion and Discussion

This chapter describes the design of a generic user interface for building operating systems by mainly focusing on the aspects of structure, access and content about a user interface. Since the concept of a generic user interface for building operating systems has not been commonly defined, this chapter begins with a proposed definition of the concept which consists of a number of conditions that the user interface needs to meet. In Section 4.2, a list of detailed system objectives was proposed for the design of such a user interface. This section discusses how these objectives are achieved by the following design in this chapter.

System Objective 1: *Provide generic data visualization and control for heterogeneous building operating systems in smart homes*

In order to realize this objective, an architecture of a generic user interface for building operating systems in smart homes has been proposed. By applying this architecture, existing building operating systems can utilize such a generic user interface as their user interface for data visualization, control and interaction with residents in smart homes. However, some certain data conversions are inevitable since the existing building operating systems usually have already been equipped with their own proprietary interfaces. The essential part of the architecture of such a generic user interface for building operating systems is a number of generic data models, which ensure that a generic user interface is able to deal with the heterogeneity of different building operating systems. To this end, this chapter comes up with a variety of data models, which are not only independent of any building operating system, but also flexible enough to be applicable to different buildings and provide users with multiple possibilities to realize various home automations.

System Objective 2: *Provide options to help building operating systems to exploit the potential of load shifting in household buildings*

The design of the generic user interface in this chapter provides three possibilities for users to specify options according to their needs in order to help building operating systems to exploit the potential of load shifting in household buildings.

Firstly, as for the conventional appliances, whose working schedules can be shifted, the household device model (cf. Figure 4.6) allows users to specify degrees of freedom for the appliances by providing a deviceDoFInfo attribute to cover the allowed starting time and the required end time of work-items of the appliances.

Secondly, this chapter introduces a calendar event service (cf. Figure 4.14) in the generic user interface. Through this service, users can plan events that will take place in their building by specifying a starting time and an end time of the events, parameters about locations where the events will take place, and target states of the devices needed by the events in order to inform building operating systems of the requirements of futuristic or periodic events in advance.

Finally, the design of the generic user interface in this chapter includes an electric vehicle model (cf. Figure 4.7). This model allows users to plan their next drive via the generic user interface by specifying various requirements including a departure time for the next drive, the distance from the target location and the minimum range that the car needs to reach.

The aforementioned information identifies the users' needs and preferences, which need to be respected by building operating systems. On the other hand, this information also meanwhile indicates the potentials of load shifting in the users' buildings. By taking this information into consideration, building operating systems can work out an optimized schedule for household devices in order to shift the domestic energy load from peak periods to off-peak periods, thus realizing demand response and saving money for users.

> **System Objective 3:** *Provide smart home users with not only a holistic but also a clear view of the energy use in their building*

In this chapter, different ways of displaying devices are proposed in order to facilitate the realization of this objective. On the one hand, household buildings are modelled as being composed of one or more floors with corresponding floor plans. Devices in a household building can not only be viewed by using a device list, but can also be visualized on the basis of a floor plan of the floors in the building, which is supposed to be a clear global way of displaying devices, and therefore provides another perspective for

users to understand the energy use in their building. On the other hand, the generic user interface designed in this chapter is able to deal with multiple energy types. The devices that are currently running in a household building are organized into different tree maps according to the particular type of the energy commodity which they are consuming or generating. These tree maps are to provide users with not only a holistic but also a clear view of global energy flows in their building.

> **System Objective 4:** *Provide smart home users with as many possibilities as possible to control appliances in their building*

In addition to the basic home automation, namely, controlling individual devices separately, the generic user interface designed in this chapter supports three aspects of advanced home automation. Firstly, users are able to group a number of devices into a customized device group. If the devices in a device group belong to the same device type, they can be controlled together via group operations. In so doing, users do not need to control the devices individually. Secondly, the generic user interface supports the creation of various scenes. By adding devices in a scene and setting target states for the devices, the scene can be triggered by the users whenever needed. After triggering a scene, the devices in the scene will automatically move into the target states that had been configured beforehand. Thirdly, events that lie in the future or those that occur regularly can be defined as calendar events on the generic user interface designed in this chapter. A calendar event can be configured by specifying parameters (i.e., temperature and humidity) for a location where the event will take place, setting target states for the devices in the event, and choosing a repeat mode for the event. Unlike device groups and scenes, a calendar event is not manually triggered by users but by the generic user interface and a building operating system. The generic user interface is responsible for activating the target states of the devices in a calendar event, and the building operating system needs to ensure that

the temperature and the humidity that have been configured for the particular location of the calendar event, are reached when the time of the event comes.

> **System Objective 5:** *Provide useful reference information to help smart home users to improve their energy awareness so as to achieve the goal of saving money*

The generic user interface designed in this chapter integrates diverse reference information in order to improve the users' energy awareness. Firstly, the concept of communities is introduced into the user interface. By providing users with the average energy use in a building community and the rankings of the users' performance within a community, users can compare the energy use and utilization of the devices in their own building with data from other buildings in the community so as to improve their awareness of energy conservation. Besides this, the proposed functional components of the generic user interface indicate that the users who are living in a household building, i.e., the residents, are allowed to not only view the real-time energy data in their building together with the current energy tariffs but also view their personal energy history as well as the historical energy data of a single device. These are also different ways of helping users to develop an awareness of whether their energy use is reasonable or not.

> **System Objective 6:** *Provide a relatively large application scope, which means the user interface is able to meet the different needs of different household buildings*

This chapter introduces three roles, namely, the administrator, the operator and the resident, to the generic user interface in order to facilitate the separation of responsibilities of users, and to deal with the diverse scenarios brought about by different circumstances. Furthermore, a generic building model (cf. Figure 4.10) is designed in order to allow for its implementa-

tion in every type of household building. For the various devices of the household buildings, a household device model (cf. Figure 4.6) is used to represent a wide range of devices with different functions and from different manufacturers. All of the data models proposed in this chapter are designed in a generic way so that the application scope for the generic user interface designed in this chapter can be expanded.

> **System Objective 7:** *Provide intuitive displays which allow users to learn to use the user interface quickly and to complete tasks by using the user interface efficiently and effectively*

This objective emphasizes a good usability of the generic user interface designed in this chapter. The usability mainly depends on the implementation of such a generic user interface, which will be described in detail in Chapter 5 and will be evaluated in Chapter 6. Nevertheless, the data models designed in this chapter indirectly facilitate a good usability of the generic user interface since the flexibility of the data models enable the generic user interface to be implemented easily and to be user-friendly. For instance, the design of the scene model (cf. Figure 4.12) along with the household device model (cf. Figure 4.6) enable scenes to be created by users in graphical ways on a user interface thereby increasing the user-friendliness of the user interface, instead of forcing users to remember technically demanding commands, which would make the user interface hardly usable by non-professionals or novice users.

5 Implementation

To evaluate the design outlined in the previous chapter in a real living environment, a prototype of the generic user interface, named Building Operating System User Interface (BOS UI), has been implemented and applied to a building operating system, the Organic Smart Home (OSH) [40], which was described in Section 2.2 of Chapter 2. Since 2009, the OSH has been deployed at the Energy Smart Home Lab of the Karlsruhe Institute of Technology (KIT). This chapter begins with an introduction to the ESHL and then explains how the user interface prototype is connected with the OSH. Furthermore, it introduces the modules used by the BOS UI and presents the detailed functionalities that have been implemented in the user interface. Part of the work in this chapter has been published in the Journal of Energy Informatics [106].

5.1 The Energy Smart Home Lab

The Energy Smart Home Lab (ESHL) was developed at the KIT by the projects MeRegioMobil[1], iZEUS[2], grid-control[3] and C/sells[4]. It demonstrates possibilities of combining in an integrated approach the areas of living (smart home), transport (electric mobility), and energy (smart grid) such that best possible use of renewable energy sources is ensured and comfort

[1] https://meregiomobil.forschung.kit.edu/
[2] http://www.izeus.de/
[3] http://projekt-grid-control.de/
[4] http://www.csells.net/de/

of living is increased at the same time [7]. Covering an area of 80 square meters, the ESHL is an advanced research and demonstration laboratory on energy management, future microgrids, integration of electric vehicles, and security as well as privacy in energy applications [44, 76]. It is built as a two bedroom apartment whose reality images are shown in Figure 5.1.

(a) The exterior of the ESHL (b) The living room of the ESHL

(c) The kitchen of the ESHL (d) The bedroom of the ESHL

Figure 5.1: The reality images of the ESHL

In addition to the rooms intended for human occupancy, the ESHL also includes a technical room where some mechanical equipment (e.g. water boiler) and electrical equipment (e.g. smart meter) is installed. Figure 5.2 shows the complete floor plan of the ESHL including both the apartment and the technical room.

The ESHL is equipped with modern household appliances which are connected to a central communication gateway via a power line, thus, no additional cable is needed for the ICT-integration of the appliances [46]. Speci-

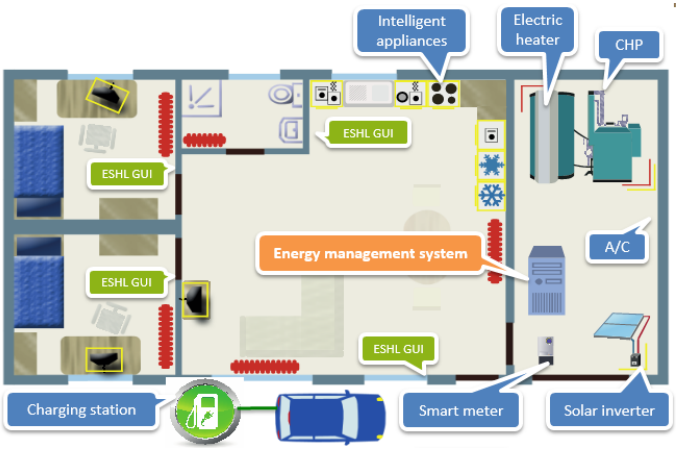


Figure 5.2: The floor plan of the ESHL [46]

fically, the ESHL is not merely a conventional apartment but rather a smart home, mainly because it is equipped with the following components:

- **Intelligent appliances**, which are able to be programmed to run from a central system. After connecting to the communication gateway of the ESHL, these intelligent appliances (e.g. washing machine and dishwasher) can be remotely configured and controlled.
- **Non-intelligent appliances**, which are simply conventional appliances (e.g. lights) that support only manual operations. In order to achieve remote control and monitoring of these appliances, additional sensors and actuators (e.g. smart plugs and relays) are provided for them in the ESHL.
- A **smart meter**, which enables the two-way communication between the meter and a utility company. This meter can therefore not only periodically provide the power consumption data of a building but can also receive tariff information from a utility.

- An **energy management system**, the OSH, which was described in Chapter 2 (Section 2.2) of this chapter. It is responsible for optimizing the schedule of appliances so as to minimize energy costs for residents [40].
- **ESHL GUI**, which is the user interface of the OSH in the ESHL. It has been further developed on the basis of the Energy Management Panel (EMP) by integrating some new engineering-related features into it. The ESHL GUI therefore can not only display the energy usage in the ESHL and help to discover user preferences for the OSH, but also support the visualization of many technical parameters in the ESHL.
- A **micro Combined Heat and Power Plant (μCHP)** which enables the co-generation of electricity and heat by burning any kind of fuel (e.g. natural gas and oil). In the ESHL, the μCHP has been extended by inserting an electrically driven heating coil into its water boiler to work as an alternative actuator to produce heat.
- **Photovoltaic (PV) panels**, which are installed on the roof of the ESHL (see Figure 5.1a). In the technical room, a solar inverter is installed in order to convert the direct current, generated by the PV panels, into alternating current that can be used locally or fed into the power grid.
- An **electric vehicle**, which is connected to the ESHL through a charging station. While the charging process of the car's battery can be controlled by the OSH to prevent grid overload, the battery can also be used as storage for electrical energy [46].

The OSH was originally designed to work as a building operating system that directly deals with devices in the ESHL. With the increasing number of challenges in the smart home environment, such as the heterogenity of devices, services, protocols and data formats, a decentralized service orien-

ted architecture, based on a message-oriented middleware (cf. Figure 5.3), was developed and has been applied to the ESHL since 2016. It is shown in this architecture that the OSH and the user interface of the ESHL are all treated as applications which interact with components from different layers, instead of dealing with devices directly so as to realize device abstraction and modularity, and further facilitate extendibility and maintainability of the system.

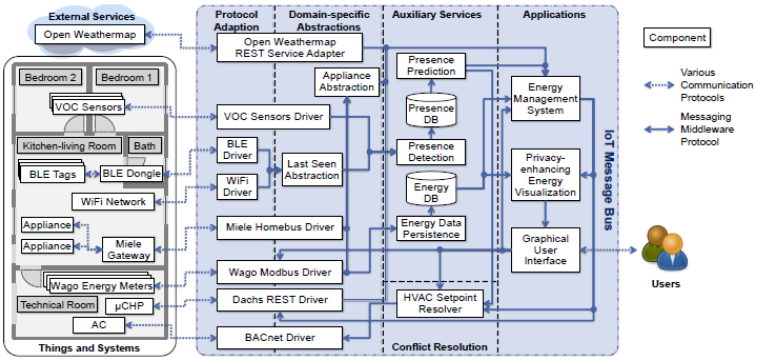


Figure 5.3: The decentralized service oriented architecture based on a message-oriented middleware in the ESHL [44]

The aforementioned architecture is composed of the following layers: a **protocol adaption layer**, which is dedicated to adapt data and functions from different protocols to standardized data formats; a **domain-specific abstraction layer**, which plays a role in converting the data after the protocol adaption into unified abstractions by means of corresponding device drivers; an **auxiliary services layer**, which helps to split system functionalities into reusable microservices; an **applications layer**, that uses these auxiliary services to provide various functionalities for users; and a **conflict resolution layer**, which is used for resolving conflicts of device operations caused by different applications. Besides this, a key element of the architec-

ture is an IoT Message Bus which supports the necessary communication schemes between all components [44].

5.2 Connection to the Energy Smart Home Lab

As described in the last section, the IoT Message Bus, which supports the necessary communication schemes between all components, is an important element in the decentralized service oriented architecture based on a message-oriented middleware. In the ESHL, the schemes Publish & Subscribe (PubSub), bulletin board, and Remote Procedure Call (RPC) are required. Because of this, the Web Application Messaging Protocol (WAMP)[5] has been selected as the communication scheme for coupling and interaction between the sensors, actuators, and services in the ESHL, since this protocol supports both PubSub and routed RPC, and bulletin boards are supported by the event history of Crossbar.io, which is its reference router implementation. [44]

The WAMP is an open standard WebSocket subprotocol that provides the two application messaging patterns, RPC and PubSub, in one unified protocol [32]. In order to achieve unified application routing for applications, WAMP provides a router which is used by the ESHL as the IoT Message Bus to realize the communication between sensors, actuators and services in the ESHL. Within the WAMP router, there are two components, a broker and a dealer, which are used as intermediaries to route PubSub events between subscribers and publishers, and RPC calls between callers and callees. As applications, the OSH and the ESHL GUI connect to the WAMP Router (cf. Figure 5.4) to interact with each other and with the sensors and actuators in the ESHL, by means of PubSub events and RPC calls.

[5] http://wamp-proto.org/

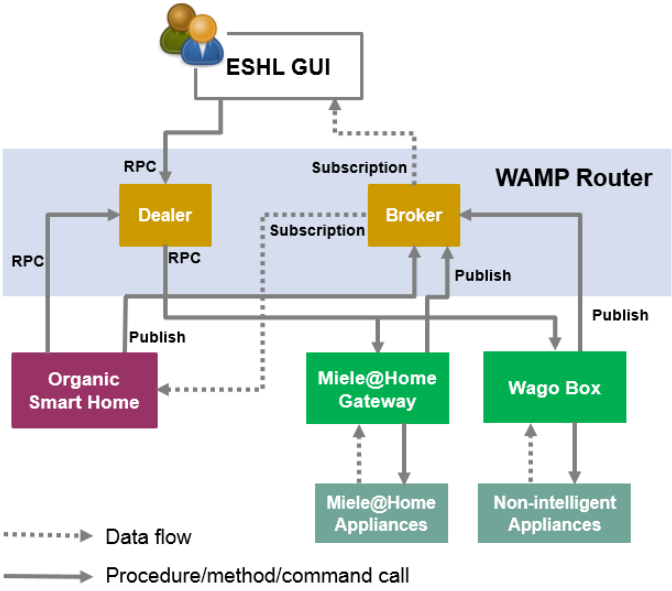


Figure 5.4: The communication among ESHL GUI, OSH, and appliances in the ESHL via a WAMP router

Generally speaking, the appliances in the ESHL can be classified into intelligent appliances and non-intelligent appliances. The intelligent appliances, which are mainly located in the kitchen of the ESHL, are from the German home appliance manufacturer, Miele. These appliances are connected to a Miele@Home gateway, which enables the Miele appliances to be networked together and to be monitored and controlled intelligently. As for the non-intelligent appliances, a so-called Wago box, a product from a German company named Wago, has been installed in the ESHL. The Wago box is able to, on the one hand, monitor the power consumption of every electrical consumer as well as each power socket in the ESHL and, on the other hand, control the state of appliances so as to realize home automation.

The measurement data from the Wago box and the Miele@Home gateway is published as different topics to the WAMP router. In addition, some other topics, such as price signals topic and weather forecast topic, are also published to the WAMP router regularly. A list of all the topics and their explanations can be found in Table 3.4. The data provided by these topics can be shared with the OSH and the ESHL GUI by subscribing to corresponding topics of interest. To this end, the WAMP router provides a Broker component, which maintains a book of subscriptions to forward the updated data of different topics to their respective subscribers. In addition to the Broker, the WAMP router includes another component, named Dealer, to take care of RPC calls. Similar to the Broker's role to PubSub, the Dealer keeps track of the procedures that have been registered in the WAMP Router. When the OSH and the ESHL GUI try to call the remote procedures in the WAMP Router, the Dealer will help to invoke them by looking up his book of all registered procedures, which records where the procedures reside and how to reach them [32]. The WAMP router in the ESHL currently supports RPC to control states of appliances and retrieve historical energy data of appliances in the ESHL as well as price signals from the utility. The RPC calls to alter states of appliances, will be forwarded to either the Miele@Home gateway or the Wago box which are responsible for controlling appliances.

Due to the function of unified application routing provided by the WAMP router, the OSH and its user interface are allowed to be loosely coupled with each other, which makes it easy to introduce the BOS UI as its new user interface, without affecting the normal operation of its original user interface, i.e., the ESHL GUI. Nevertheless, since the BOS UI is designed as a generic user interface rather than a special user interface for the OSH, a few extra components need to be added to the system in order to complete the connection with the WAMP router and realize the communication between the BOS UI and other entities in the ESHL. The relationship between these components, the WAMP router and the BOS UI is shown in Figure 5.5.

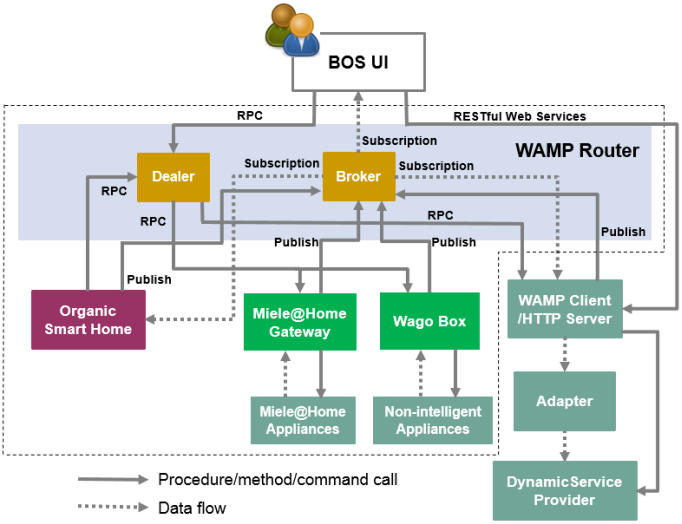


Figure 5.5: The relationship between the newly added components, the WAMP router and the BOS UI

The components that need to be added to the system will now be specified in this section, together with an explanation of their functions.

The **WAMP Client/HTTP Server** is a component that includes two functions. Firstly, it acts as a WAMP client, which communicates directly with the WAMP router. To this end, a few parameters need to be provided in the initialization stage to configure the client in order to create a connection to the WAMP router. After the establishment of the connection, the WAMP client subscribes the topics that the BOS UI is interested in from the WAMP router, and will then keep receiving the updated information of these subscribed topics. The WAMP client also publishes its own topics to the WAMP router which can then be subscribed to by the BOS UI so as to provide it with extensive information, such as, for example, the two following topics:

- Global energy data topic which reflects some building-level energy parameters, such as the ESHL's overall power, voltage, and frequency, etc.
- Power consuming devices topic which reports a list of devices that are, at that instant, either consuming (with a positive value) or generating power (with a negative value).

What is more, as a WAMP client, the component also registers procedures at the WAMP router. The remote procedures will be called by the BOS UI in order to get specific information it needs especially when variables are required for retrieving this information. For instance, when the BOS UI calls a procedure named "get_a_device" by passing a device identifier as a parameter, it will receive an instance of a certain device as a return from which various property values about the device can be derived. Since the biggest advantage of WAMP is the fact that it allows a web-based user interface to talk to its back-end system in real-time, the WAMP client in this case is mainly responsible for helping the BOS UI to get the real-time states of the OSH and the devices in the ESHL.

In addition to acting as a WAMP client, the WAMP Client/HTTP Server component also acts as an HTTP server for RESTful Web Services that will be accessed by the BOS UI. The reason for the introduction of Web Services is that a lot of information in the ESHL, such as building locations and configurations, which usually only need to be set once, will not be updated dynamically or frequently in the future. In order to enable access and operation of this static data in the ESHL, a number of RESTful web services, which take advantage of the existing HTTP protocol, have been implemented to bridge the gap. The Application Programming Interfaces (APIs) of some of the important services and their descriptions can be found in Table 5.2.

Table 5.1: Some important RESTful APIs designed for the BOS UI

Resource	Methods	Descriptions
Floor	• GET /floors	Return a list of floors in the building
	• GET /floors/{floorId}	Return a floor with an id of floorId
	• POST /floors	Create a floor in the building
	• PUT /floors/{floorId}	Update a floor with an id of floorId
	• DELETE /floors/{floorId}	Delete a floor with an id of floorId
Location	• GET /locations/{floorId}	Return a list of locations in the floor with an id of floorId
	• GET /locations/{floorId}/{locationId}	Return a location with id = locationId in the floor with an id of floorId
	• POST /locations/{floorId}	Create a new location in the floor with an id of floorId
	• PUT /locations/{floorId}/{locationId}	Update a location with an id of locationId in the floor with an id of floorId
	• DELETE /locations/{floorId}/{locationId}	Delete a location with an id of locationId in the floor with an id of floorId
User	• GET /users/verification (QueryParam1: username, QueryParam2: password)	Verify if the given username and password are correct. If correct then return the user
	• GET /users/residents	Get all of the residents in the building
	• POST /users	Create a user in the system
	• PUT /users/{username}	Update a user with an id of username
	• DELETE /users/{username}	Delete a user with an id of username
Scene	• GET /scenes/{username}	Return the scenes that have been created by the resident with an id of username
	• POST /scenes/{username}	Create a scene for the resident with an id of username
	• PUT /scenes/{username}/{sceneName}	Update the scene with an id of sceneNmae for the resident with an id of username
	• DELETE /scenes/{username}/{sceneName}	Delete the scene with an id of sceneNmae for the resident with an id of username

Continued on next page

Table 5.1 – continued from previous page

Resource	Methods	Descriptions
Device Group	• GET /deviceGroups/{username}	Get device groups that have been created by the resident with an id of username
	• POST /deviceGroups/{username}	Create a device group for the resident with an id of username
	• PUT /deviceGroups/{username}/{groupName}	Update the device group with an id of groupName for the resident with an id of username
	• DELETE /deviceGroups/{username}/{groupName}	Delete the device group with an id of groupName for the resident with an id of username
Log	• GET /logs/{username}	Return a list of logs which are related to the resident with an id of username
	• POST /logs/{username}	Create a log for the resident with an id of username
	• DELETE /logs/{username} (QueryParam1: startTime, QueryParam2: endTime)	Clear the logs for the resident with an id of username from startTime to endTime

As mentioned above, the WAMP client receives data from the WAMP router by subscribing topics that it is interested in. The format of the data is, however, defined by the entities who publish the topics and therefore cannot be directly used by the BOS UI. As a solution, this data is further sent to another component, named **Adapter**, which receives the original data published by different entities in the ESHL as the input, and then converts this data into unified data models (cf. Section 4.6) that can be consumed by the BOS UI. To this end, the Adapter needs to know both the structures of the data from the WAMP router and of the unified data models used by the BOS UI. This makes the Adapter an intermediary component, since the services it provides will further be called by the **DynamicServiceProvider** component in order to realize the data conversion. The DynamicServiceProvider is a component that provides all kinds of integrated services (cf. Table 5.2) for

the BOS UI by collecting the dynamic data from the WAMP client, such as varying power values of devices, and converting them into standard data models with the help of the Adapter. These services will be used by the WAMP client to publish new topics or register new procedures to the WAMP Router.

Table 5.2: Integrated services provided by the DynamicServiceProvider component

Service Name	**Parameters**	**Descriptions**
getADevice	deviceId, language	Return a device with an id of deviceId. The returned object is expressed with the given language (English, German or Chinese, similarly hereinafter).
getAllDevices	language	Return all devices in the building. The returned objects are expressed with the given language.
getPoweringConsumingDevices	language	Return all devices in the building that are consuming (positive value) or generating (negative value) power. The returned objects are expressed with the given language.
getBuildingGlobalEnergyData	language	Return global energy data (e.g. total power, voltage, frequency) of the building. The returned objects are expressed with the given language.

In conclusion, with the aid of the aforementioned components, the BOS UI is able to communicate with the OSH and with sensors and actuators of a variety of devices in the ESHL. As for dynamically updated data (e.g. device states) or different historical energy data of the ESHL, the BOS UI can retrieve this by either subscribing to certain topics which have been published to the WAMP router or calling the remote procedures that have been registered in the WAMP router. Data which is not updated dynamically or frequently in the ESHL, can be accessed and updated by the BOS UI by means of a number of RESTful Web Services.

5.3 Modules

The BOS UI was developed on the basis of Fuse[6], which is an AngularJS template that uses Angular Material library. AngularJS[7] is a client-side JavaScript web framework for creating single-page web applications based on a Model-View-ViewModel (MVVM) design pattern. The difference between the MVVM design pattern and the commonly used Model-View-Controller (MVC) design pattern is that the MVC pattern supports only one-way data-binding while the MVVM enables two-way data-binding. Data binding in AngularJS is the synchronization between the model and the view, which means, when data in the model changes, the view reflects the change, and when data in the view changes, the model is updated as well [2]. The feature of two-way data-binding supported by AngularJS can help applications to improve their user experience since there is a direct visual feedback of the user's actions on the applications.

In order to keep the code clean, AngularJS supports a modular approach. A module in AngularJS is a feature or a container for the different parts of an application. It allows grouping of related features so as to make reusable and maintainable components for applications. In the BOS UI, a variety of modules (cf. Table 5.3) have been created in order to cover all the features proposed in Section 4.7 of the previous chapter. These modules are implemented based on the generic data models in Chapter 4. Nevertheless, some of the modules are not completely generic yet, since the historical energy data series used by the BOS UI for creating various charts are currently still from the result of directly accessing data records stored in the database of the ESHL via topic subscriptions. In order to extend these modules to other smart homes, the way of extracting information from historical energy data series might need to be adjusted due to different data formats in different

[6] http://fusetheme.com/angularjs/
[7] https://angularjs.org/

smart homes. Considering the fact that historical energy data normally consist of a large amount of data records (depending on a certain data resolution and time period), it would be time intensive and also not practical to convert each of the data records to a unified data structure. Therefore, unified data structures are not designed for historical energy data series in this thesis.

Table 5.3: The modules that have been created for the BOS UI

core	**login**	**administrator**	**operator**
navigation		• administrator.locations	• operator.residents
toolbar		• administrator.devices	• operator.optimization-goals
quick-panel		• administrator.floor-plan	• operator.building-history
			• operator.community
			• operator.utility
resident		• resident.energy-history	• resident.event-logger
• resident.energy-overview		• resident.energy-history.personal-history	• resident.electric-vehicle
• resident.device-overview		• resident.energy-history.device-history	• resident.life-assistant
• resident.device-overview.device-list		• resident.device-groups	• resident.life-assistant.notes
• resident.device-overview.floor-plan		• resident.scenes	• resident.life-assistant.weather-forecast
• resident.energy-prediction		• resident.calendar	

The modules in Table 5.3 can generally be divided into three categories. The first category has the modules for the UI framework, which are provided by the Fuse template. It includes a core module which provides core functionalities and services for the UI framework, a navigation module which is responsible for rendering the side navigation menu, the toolbar module which adds a toolbar component to the UI framework, and a quick-panel module which is used to add an extra side panel, called Quick Panel, to the UI framework in order to extend the content space for the BOS UI. The second category is the login module which is designed for controlling the users' access to the BOS UI by authenticating their identity. The third category consists of a number of modules to realize the functionalities of the

three roles in the BOS UI. There are three main modules, an administrator module, an operator module, and a resident module, which correspond to the three roles, respectively. Each of the main modules is composed of a number of sub-modules so that the code of different features of the BOS UI can be organized into separate logical containers. By taking advantage of sub-modules, new functionalities can easily be introduced (i.e., load corresponding sub-modules) to the BOS UI and, at the same time, undesired features (i.e., unload corresponding sub-modules) can be flexibly removed from the BOS UI. All these operations on sub-modules can be performed under the premise of keeping the existing code base unaltered.

5.4 Functional Demonstration

This section demonstrates the detailed functionalities that have been implemented in the BOS UI. In view of the fact that the users of the BOS UI are classified into three roles, namely, the administrator, the operator and the resident, the BOS UI also offers three different views for these three roles. The implementation of the BOS UI will be introduced in this section from the perspective of the three roles, respectively.

5.4.1 Administrator

The major responsibility of the administrator is to configure the building. The user interface for the administrator consists of three menu items. The first item is "Manage Locations" where the administrator can add floors to the building model in the BOS UI and define locations for the floors. Figure 5.6 shows an example of a floor that has been configured for the ESHL. The administrator can also add a new location to the floor or add another floor to the building model in the BOS UI if the physical household building is multi-storey. The second menu item is "Assign Device to Floor" in which

the administrator can assign the devices in the building to their proper locations, which were defined in the first menu item.

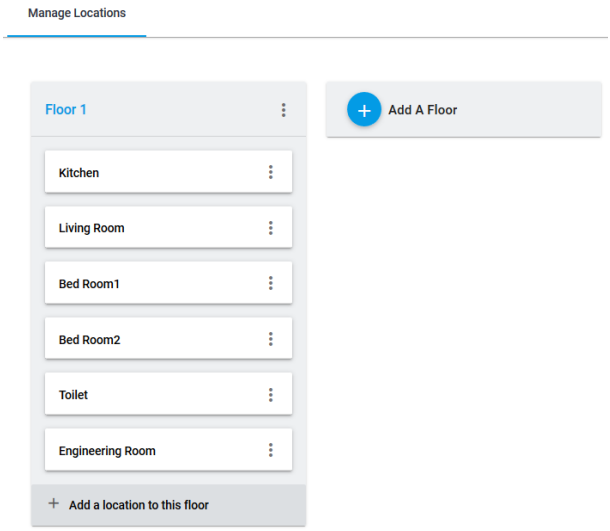


Figure 5.6: BOS UI for the administrator: the location management

The last menu item is "Add Device to Floor Plan" (cf. Figure 5.7). The content under this menu item consists of a list of tabs, which represent the floors that have been previously added to the building via the first menu item, namely, "Manage Locations". By entering each of the tabs, the administrator can upload the corresponding floor plan image for that floor. According to the design of the data model of the Floor (cf. Figure 4.9), the floor plan of a building and the devices in the building can be combined together in order to create an intuitive overview for users. For this purpose, a device panel is provided to display all the devices that can be added to the floor plan. The device images on this device panel can be dragged to the positions on the floor plan corresponding to their physical locations in the building. The size of the devices on the floor plan can also be changed by dragging any of the

four round circles around the devices. After the configuration, the floor plan and all the devices on it can be accessed by the residents in their views.

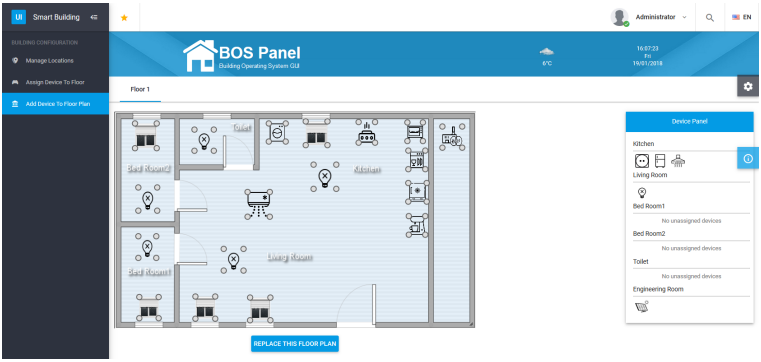


Figure 5.7: BOS UI for the administrator: adding devices to a floor plan

5.4.2 Operator

The operator is responsible for managing residents and paying the energy bills of a building. Figure 5.8 shows a content page in which the operator can add a new resident to the building. Three steps are needed in order to complete the addition of a resident. Firstly, the operator needs to specify the resident's personal information including a username, an initial password, a phone number, an email address, etc. Secondly, proper permissions need to be assigned to the resident. Every permission consists of a number of devices that are allowed to be used by the resident together with the permissible operations corresponding to these devices. The optional operations include "view device general information", "view device channel information", "control device", and "set degree of freedom". These operations have already been explained in Section 4.5. It should be noted that, electric vehicles in this thesis are not considered to belong to household devices. Instead, electric vehicles have their own data model (cf. Figure 4.7), which has been

explained in Chapter 4. For an electric vehicle, optional operations include only "view device general information", "view device channel information", and "set degree of freedom". These operations correspond to viewing the car's general information, viewing battery information of the car and setting next drive for the car, respectively. Finally, the operator needs to check all the information that was configured in the previous two steps and then submit the information to the server in order to complete the creation of the account for the resident.

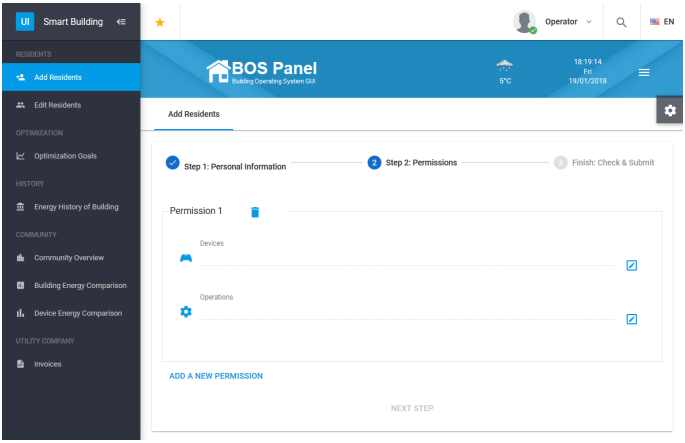


Figure 5.8: BOS UI for the operator: adding a resident

Besides this, since the operator is responsible for paying the energy bills of the building, he is allowed to set optimization goals for the building (cf. Figure 5.9). In the BOS UI, four optimization goals are provided, namely, minimal costs, minimal energy consumption, minimal CO_2 emissions and maximal self-consumption of renewable generation. Each of these goals is presented with a slider with which the operator can set a particular weighting (within the range of 0 to 1) for the goal to indicate its relative importance. Competing goals set by the operator are supposed to be balanced according to their weightings via the multi-objective optimization methods of the buil-

ding operating system. It is noteworthy that the optimization goals formulated in this thesis are neither predefined nor achieved by the BOS UI. Rather than the BOS UI, it is a building operating system that supports a list of optimization goals, and that makes the achievement of these goals transparent to the household residents. Optimization of in-house energy consumption itself has to be realized by combining the optimization algorithm used in the building operating system and the residents through adapting their behavior according to the BOI UI suggestions. As a user interface, the BOS UI is only displaying the goals provided by the building operating system, which further justifies the requirement that the BOS UI is not coupled with a specific building operating system.

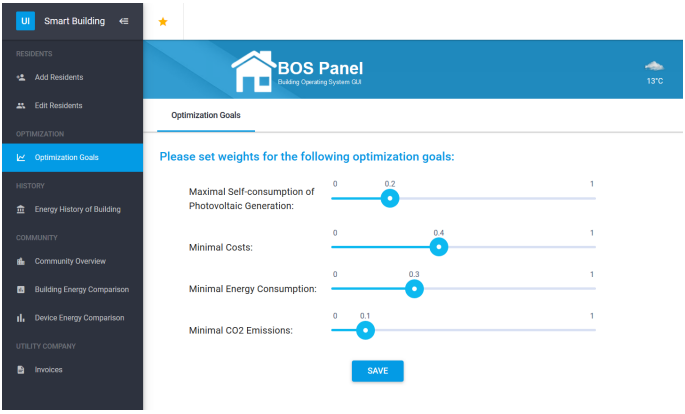


Figure 5.9: BOS UI for the operator: setting optimization goals for a building

In order to get a clear overview of the building's historical energy use, the BOS UI enables the operator to check the energy history of the building (cf. Figure 5.10). The operator merely needs to specify a start time and an end time for the history records that he would like to see displayed on the BOS UI. The type of the historical data that may be viewed includes the building power consumption, the building power generation, the building net power use, the electricity price, the load limits, the PV feed-in price and the µCHP

feed-in price. The operator can decide which of the data they would like to have displayed on the diagram by checking or unchecking the options corresponding to the data type on the top of the selection panel.

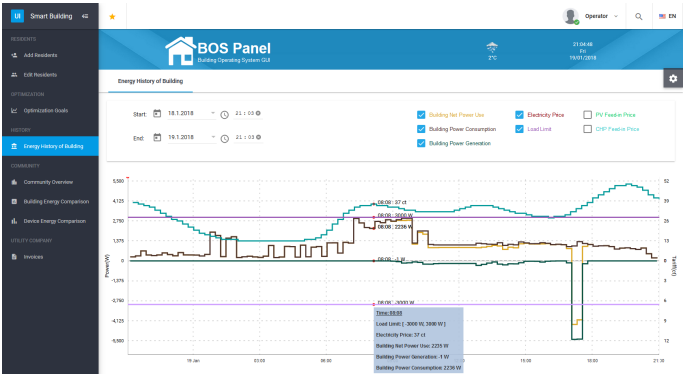


Figure 5.10: BOS UI for the operator: the energy history of the building

A novel feature that has been implemented in the BOS UI is the integration of community services, which have, up until now, only been mentioned as in some research papers as future concepts (e.g. [1] and [78]). Since this feature is currently not supported by the OSH, communities in the BOS UI are a preliminary design feature which makes provision for the future. Via the BOS UI, the operator will be able to get an overview of all the communities that they have either joined or not joined (cf. Figure 5.11). According to the design of the data model of the community described in Section 4.6 (cf. Figure 4.17), the basic information pertaining to a community, e.g. the building types which are part of the community (household building, office building, industrial building or commercial building) and the number of buildings for each building type in the community, are predefined by the data model. Some more detailed information that might be beneficial to the operator to give them a clearer picture of the community could be provided by the community service provider. The BOS UI will be responsible for

displaying this information for the operator. By joining a community, the operator will be able to enjoy the services rendered by the community, e.g. comparing the energy consumption in his building with the average value in the community. To this end, the operator will be required to agree to share the meter data of his building with the communities. Considering privacy concerns, the operator can define the time granularity (which could be 15 minutes, 30 minutes, 1 hour or two hours in the current version of the BOS UI) for the smart meter in the building, thus specifying the frequency with which it is to deliver the load profiles to the community. After joining a community, the building is, by default, first of all disconnected from the community, i.e., the building will not start transferring the meter data at the specified interval to the community until the operator manually connects the building to the community. Over and above this, the operator can choose to disconnect from the community at any time or to even 'quit' the community.

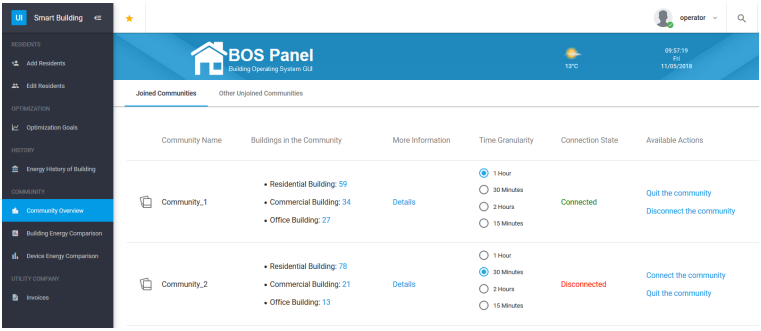


Figure 5.11: BOS UI for the operator: the community overview

The advantages of the feature offered by the BOS UI, of being able to join a community can be discussed from two points of view. Firstly, the operator can compare the energy use in the building with that of the community during a particular month. The content of the comparison is classified into four types in the BOS UI, namely, energy consumption in the building, energy generation in the building, net energy use in the building, and greenhouse

gas emission from the building. Each type of content can, in the current version of the BOS UI, be compared by means of using one of the following two modes, namely, per person or per square meter. For instance, the operator can choose to compare the average energy consumption per person in his building in December 2017 with other same type buildings in a community, named "Community_1". The comparison report (cf. Figure 5.12) illustrates the average energy consumption per person (KWH/person) and the average energy cost per person (Euro/person) in the building as well as the counterparts in the community along with the building's corresponding rankings in this community. Furthermore, the two bar charts in the report display the number of buildings that consumed the same averages of 'energy consumption per person' (KWH/person) and the number of buildings that spent the same averages of 'energy cost per person' (Euro/person) in the community, respectively. The report can be downloaded in the form of a PDF file or can be shared on various popular social networks by the operator, thus fulfilling the goal of enabling the sharing of the energy achievement in the building with friends on social networks (e.g. facebook, twitter, etc.).

In addition to the energy comparison relating to the whole building, the operator can also choose to compare the energy use of a specific device in his building with data from other buildings in a community. To this end, the operator needs to select a certain month for the comparison, a specific device which is to be compared, a community in which the device will be compared, and a comparison mode. On the BOS UI, a device in a building can currently be compared in one of the following four modes: power use per person (KWH/person), energy cost per person (Euro/person), power use per usage (KWH/usage) and, energy cost per usage (Euro/usage). Utilizing the comparison capability of the BOS UI, the operator is able to analyze whether the devices in his building are energy efficient and whether they are being used by the residents in the building in an appropriate manner, by comparing them with the use of devices of the same type in the community.

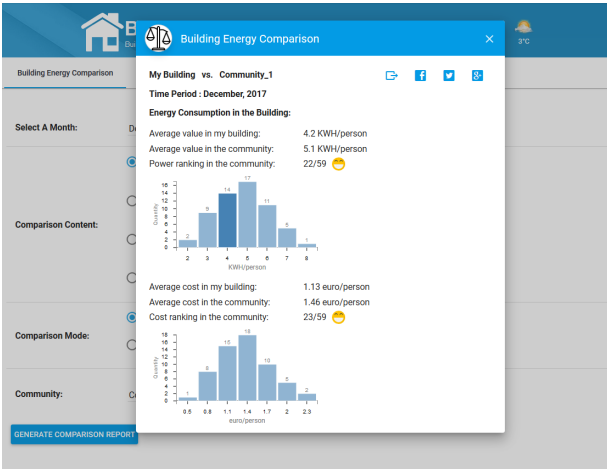


Figure 5.12: BOS UI for the operator: building energy comparison in a community

Furthermore, the BOS UI is able to collect the invoices which are sent regularly by the utility company and show them to the operator. Various other utility functions of this type could be implemented in a future version of the BOS UI.

5.4.3 Resident

In addition to the administrator and the operator, another role in the BOS UI is the resident, which represents the users who live in the building and use the devices on a daily basis. The BOS UI not only provides residents with all encompassing and intuitive views for visualizing their energy use in their home but also offers various functionalities to support advanced home automation in order to improve the residents' comfort and facilitate their daily lives. Figure 5.13 shows the content page of the energy overview provided by the BOS UI for the role of resident. The energy overview page

is made up of two parts, namely, a main panel in the middle and a side panel on the right.

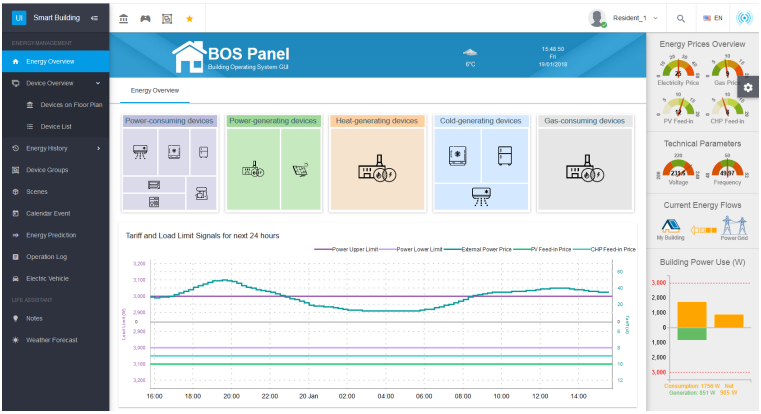


Figure 5.13: BOS UI for the resident: energy overview

On the top of the main panel (middle), there are five tree-map widgets of different colors, which display the devices that are consuming power, the devices that are generating power, the devices that are generating heat, the devices that are generating cold and the devices that are consuming gas, respectively. This tree-map is the implementation of the design of the data structure for energy overview described in Section 4.6 (cf. Figure 4.18) of the previous chapter. The tree-map is organized into the tree structure. It consists of a number of tiles which are equivalent to the nodes of a tree. Each tile represents a device group or a device in the building. The advantage of using the tree-map widget is that the size of each tile can be set to be proportional to the amount of energy that the corresponding device is consuming or producing, so that residents can intuitively identify which devices are comparatively bigger energy consumers or generators. When the mouse hovers over a tile, some basic information about the device (e.g. power and location, etc.) is displayed. By clicking on a tile, residents can view the detailed information about the device and control it. Device groups are pre-

sented as a group of tiles that are integrated together and can be spread out or folded up by clicking on the tiles in the group. Because the tree-map widgets are composed of a number of dynamically resizable tiles, they can give residents a holistic and clear view of the energy use of the different devices in their building.

Below the tree-map widgets on the main panel, there is a chart which shows the tariff (including external electricity price, PV feed-in price and μCHP feed-in price) and load limit (including load upper limit and load lower limit) signals for the next twenty-four hours. The predicted signals coming from the Distribution System Operator (DSO) and the energy provider are to help residents to rationally improve their use of the devices in their building with regard to energy efficiency, load shifting, etc.

The widgets in the main panel (middle) are flexible so that residents can change their size and/or move them to different positions according to personal preferences. For instance, Figure 5.14 shows a different layout in which the widgets have been resized and organized in a different manner, to allow for different preferences of residents.

Next to the main panel in the middle of the energy overview page is a side panel (cf. Figure 5.15a), which displays some real-time auxiliary information, including the current energy prices (i.e., external electricity price, gas price, PV feed-in price and μCHP feed-in price), the instantaneous technical parameters in the building (i.e., voltage and frequency), the current direction of the energy flow in the building and the global power use information of the building. Since the energy prices and technical parameters in the building are continuously changing, this data is presented as gauges, similar to the speedometer gauges for cars. The gauges on the BOS UI are combined with different colors in order to indicate different price levels and security information concerning the power use in the building. This makes it more intuitive for residents to understand the meaning behind the numbers

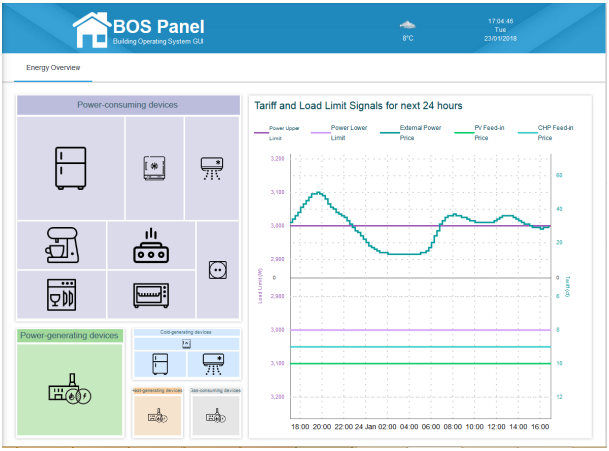
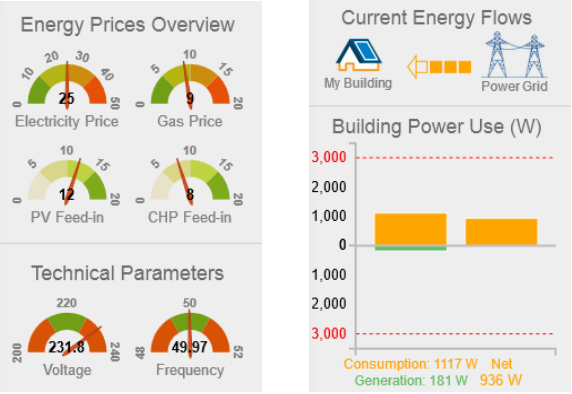


Figure 5.14: BOS UI for the resident: resized and rearranged widgets for the energy overview

quickly, so that they will be able to adjust their behavior in time and thus enable a quick response on their part.

.

Below the gauges in the side panel (cf. Figure 5.15b), residents can view the current energy flow of the building, whose direction is either from the building to the power grid or the other way around, depending on the relationship between the amount of energy that is being consumed and the amount of the energy that is being generated in the building. The actual relationship is illustrated by the bar-chart widget at the bottom of the side panel. From this widget, residents can determine exactly the global power consumption/generation in their building. The orange bar above the X-axis represents the current power consumption in the building, and the green bar under the X-axis represents the current power generation in the building. The third bar on the right represents the net power use in the building, which also determines the direction of the energy flow.



(a) Current energy prices and technical parameters in a building

(b) The current energy flow and power use in a building

Figure 5.15: BOS UI for the resident: real-time auxiliary information on the side panel of the energy overview page

The daily life of residents is driven by the use of household devices. Therefore, the BOS UI provides residents with different perspectives for viewing and controlling the devices in their building in order to achieve home automation in a more user-friendly way. One perspective is to display devices based on a floor plan. The BOS UI is initially configured by the administrator who is responsible for uploading floor plans for the floors of the building and placing devices on the corresponding floor plans according to their physical location in the building. After the configuration is complete, residents can see the floor plan with their devices (cf. Figure 5.16) in their views after logging into the BOS UI.

Since the operator assigns residents with different permissions for accessing devices, only those devices that the residents have been given permissions to access will be visible on the floor plan. In accordance with the design of the Floor model (cf. Figure 4.9) described in the previous chapter, every

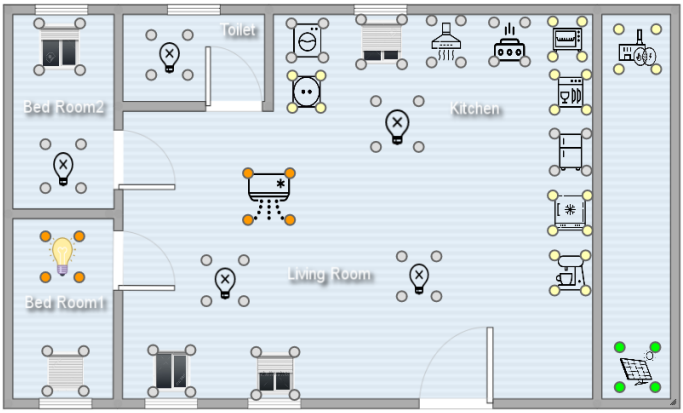


Figure 5.16: BOS UI for the resident: overview of devices on the floor plan

device on the floor plan is surrounded by four small circles, whose color indicates the current running state of the device. The four colors used by the BOS UI together with their corresponding meanings are:

- Gray: the device is off and its power consumption is zero.
- Yellow: the device is idle or standby and consuming very little power (which is usually a few Watts).
- Orange: the device is running and consuming power.
- Green: the device is generating electricity.

The allocation of different colors to the device images on the floor plan, provides residents with an intuitive, clear and holistic overview of the running states of the devices in their building. Devices can also be controlled by 'clicking on' the device images on the floor plan. Figure 5.17 shows the pop-up dialog box which appears when 'clicking on' the washing machine of the floor plan. Besides looking at the basic information of the washing machine, residents can also set a degree of freedom by specifying the allowed starting time and the required end time for their laundry. Over and

above this, there is a "Turn On" button which gives residents full control by allowing them to turn on their washing machine at any point in time. All the information which is displayed in the dialog box comes from the building operating system. The latter provides this information by instantiating the generic HouseholdDevice model (cf. Figure 4.6). Thus the BOS UI itself does not define the content of the dialog box for the device, but simply retrieves the information from the HouseholdDevice instance provided by the building operating system, and then displays the attributes and their corresponding values in the dialog box. This guarantees the generality and scalability of the user interface.

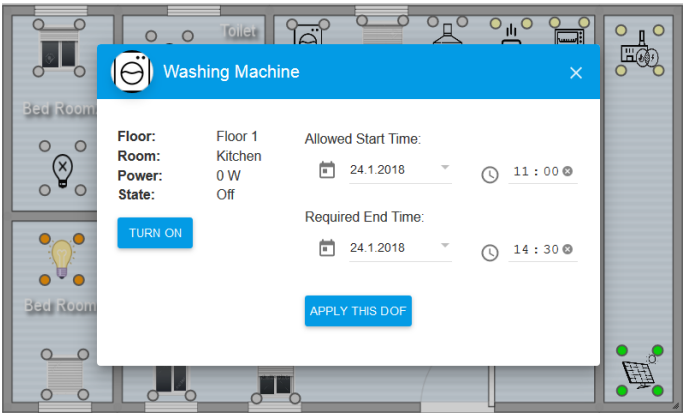


Figure 5.17: BOS UI for the resident: detailed information of a washing machine on the floor plan

In addition to displaying the devices on the floor plan, the BOS UI also allows residents to view their devices in the form of a device list. The devices in the building are grouped into different lists according to different criteria, such as locations, energy types, and customized device groups etc. The alternative modes for viewing devices, which are provided by the BOS UI, enable residents to choose their preferred mode, in order to be able to get a quick overview of the energy use of the devices in their building.

Home automation is a basic function of a smart home. The BOS UI not only implements the basic automation, which enables residents to control devices in the building individually, but also supports different aspects of advanced home automation, namely, device groups, scenes and calendar events. These three types of advanced home automation are implemented in the BOS UI, based on the design of the device group model, the scene model and the calendar event model introduced in the previous chapter. Firstly, the BOS UI allows residents to customize their own device groups. By combining a number of devices into one group, residents are able to have centralized control over the devices in the group. Figure 5.18 shows two customized device groups in the BOS UI, namely, one containing blinds and one containing lights. The devices in the group can either be controlled separately by clicking on the corresponding items in the group panel or can be controlled together as one, by clicking on the group menu on which a global controller is provided to set a state for the entire group of devices.

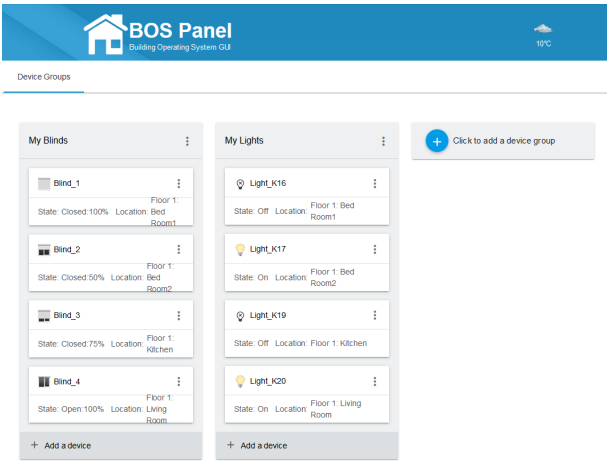


Figure 5.18: BOS UI for the resident: customized device groups

Secondly, the BOS UI implements advanced home automation based on scenes. Residents can create different scenes according to their needs and add corresponding devices to the scenes. Due to the flexible design of the HouseholdDevice model (cf. Figure 4.6), the BOS UI is able to extract the available actions for the individual devices in the scene, and display the widgets corresponding to these actions for residents so that they can specify the target states for these devices. The command strings corresponding to the target states of the devices will be saved by the BOS UI. When the resident triggers a scene, the BOS UI will iterate through the devices in the scene, obtain the command strings for their target states and send them to the building operating system so as to realize the control over the devices in the scene. Figure 5.19 shows an exemplary scene named "Sleep". Five devices, including two lights whose target state is off, two blinds whose target state is 100% closed, and an air conditioner whose target temperature is 23°C, have been added to this scene. The target states of the devices are specified by adjusting the widgets in the column of "Available Actions". The scenes that have been created can be triggered at any time by the resident.

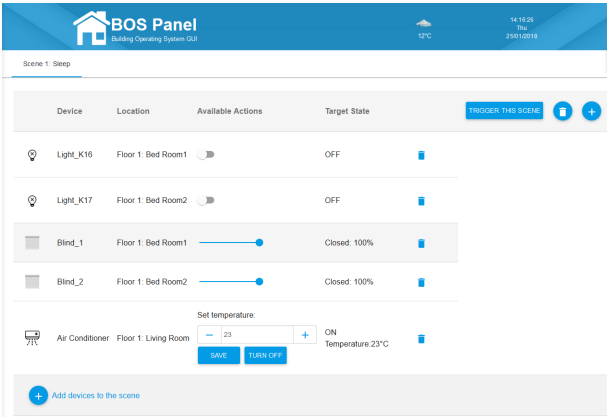


Figure 5.19: BOS UI for the resident: an exemplary scene

Thirdly, apart from device groups and scenes, the BOS UI integrates a calendar component which not only provides the basic functions of a calendar but also allows residents to add events to facilitate advanced home automation. The calendar events can be added by configuring devices or locations or both. Figure 5.20 shows the dialog box for adding a calendar event, which has been divided into four parts, namely, the title of the event, the configuration for the event start, the configuration for the event end and, the repeat mode. In order to add an event, the resident firstly needs to enter a title for the event and specify a starting time. After that, he can choose to 'set location' or 'set device(s)' for the event. By choosing 'set location', he can select a location in the building where the event is to take place and configure the temperature and/or humidity for that location. The resident can also choose to 'set device(s)' for the event by specifying target states of certain devices which are favorable for the event. After the configuration of the event start, the resident may configure states of the location and/or devices at the end of the event. Finally, the repetition mode of the event can be configured. Non-recurring events, which will happen only once, can be set not to repeat. If events are expected to be repeated, they can be set to repeat, either every day or every week over a period of time.

For example, the following two calendar events could be added to the BOS UI:

Calendar event 1: Blinds automation. At 8:00 am, open all blinds in the building and close them at 7:00 pm. Repeat the event every day from 01.02.2018 to 01.05.2018.

Calendar event 2: A meeting event. The event will be held in the meeting room from 4:00 pm - 5:00 pm, 01.03.2018. During the event, the temperature and the humidity of the meeting room should be 22°C and 60%, respectively. Switch on the lights and roll up the blinds for the event. When the event is over, disable the temperature and humidity settings, switch off the lights and roll down the blinds.

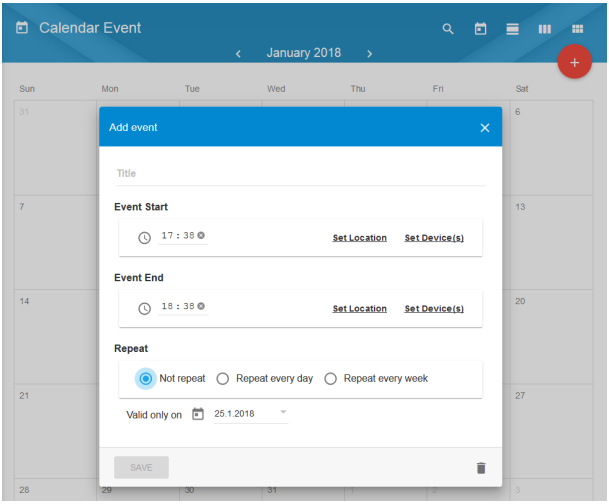


Figure 5.20: BOS UI for the resident: the dialog box of adding a calendar event

After configuration, the events will be marked on the particular building calendar of the BOS UI, which will ensure that the devices in each event will reach their target states set by the resident. The device and location settings defined by the resident will be sent by the BOS UI to the building operating system so that they can be integrated into the global energy optimization for the building.

In addition to normal household devices, the integration of electric vehicles (cf. Figure 5.21) is also supported by the BOS UI. Having the capability for storing energy and for bidirectional utilization, the electric vehicle can connect to the building and be used by the building operating system as a controllable load or a mobile storage. This may only be done on the premise of meeting the needs of the owner of the electric vehicle. Therefore, apart from displaying the current charging state and some external signals, the BOS UI allows the owner of the electric vehicle to plan his next journey by specifying a number of parameters, including the departure date and time,

the distance of travel for the next trip, and the minimum range that the car needs to be able to cover. The requirements specified by the resident have to be respected by the building operating system. Together with the Time-of-Use tariff, load limits, and the electric vehicle's maximum charging power, the charging flexibility for the electric vehicle can be determined. By exploiting the flexibility of charging demand, the building operating system is supposed to be able to devise a charging schedule for the electric vehicle as the result of the optimization algorithm of in-house energy use supported by the building operating system. Nevertheless, the BOS UI, can also control the electric vehicle to start charging immediately without taking the optimization strategy of the building operating system into account. The purpose of having this option on the BOS UI is to give residents a sense of control over their electric vehicles.

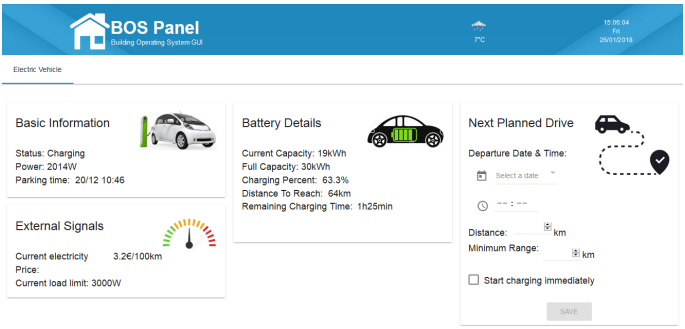


Figure 5.21: BOS UI for the resident: the configuration of the electric vehicle

Residents can not only control devices in a variety of ways via the BOS UI, but can also monitor the energy history of the building. Since the residents of a building have different permissions for accessing devices, for the sake of privacy protection, the BOS UI allows residents to only view the history of their personal energy use as well as the energy use history of individual devices for which they have obtained permissions. Figure 5.22 illustrates a resident's historical energy use of the coffee system in the ESHL,

on 26th January, 2018. At the top of the display is the selection panel where the resident can choose a historical date, a device that he has permissions to use, and a community in which he would like to make comparisons. The device history data consists of two parts. One part is the day's overall data, and the other part is the real-time energy consumption/generation diagram. The day's overall data is composed of the total energy data, i.e., power consumption/generation of the device in the building on the selected day and its corresponding cost/profit, together with the average energy consumption/generation and cost/profit of the same type of device in a community, as well as the ranking of the data of this device in the community. Another part of the device history is the real-time energy consumption/generation diagram, which illustrates the real-time energy consumption or generation of the device during the day as well as the corresponding external energy signals (including a load limit signal and an external electricity price signal) that can be used as references for the use of the device.

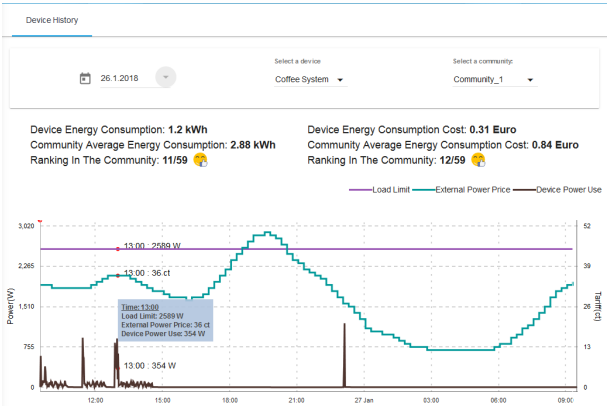


Figure 5.22: BOS UI for the resident: the historical energy use of a single device

In addition to monitoring the historical energy use of a single device, a resident is also able to view the history of his personal energy use, which is the sum of all the power consumption/generation of the devices for which he has

permissions in the building. Similar to the energy history of a single device, the personal energy history is also composed of two parts. One part is for the resident's overall energy use (including energy consumption/generation and corresponding cost/profit along with their average values of a community and the ranking in the community), and another part is the real-time energy use (including a load limit signal and an external energy price signal) during the selected day.

In order to support residents in adjusting and optimizing their energy use in the future, the BOS UI is able to display a prediction, over the next 24 hours, of the energy use of their building together with the external energy signals (cf. Figure 5.23). The types of data available for the 'predicted energy use' include the base load power and the net load power of the building, the energy generation from the photovoltaic (PV) and the energy generation/consumption from the μCHP. The future external energy signals displayed on the BOS UI are the external electricity price, the load limits (including both the load upper limit and the load lower limit), and the electricity feed-in prices of the PV and the μCHP, respectively. In order to facilitate a comparison, the different types of predicted energy data and the external signals are displayed in one diagram, but the resident can choose to display or hide these items on the diagram by 'checking' or 'unchecking' the corresponding options on the top panel.

What is more, the BOS UI provides residents with a device operation log (cf. Figure 5.24), which records a chronological documentation of how the devices in the building have been controlled. Detailed information includes the device name, the location of the device, the time that device was operated, the command executed by the device, the operation mode and the executor who sent the command to the device. The operation mode describes how the devices were controlled. In the BOS UI, there are five operation modes which are defined as follows:

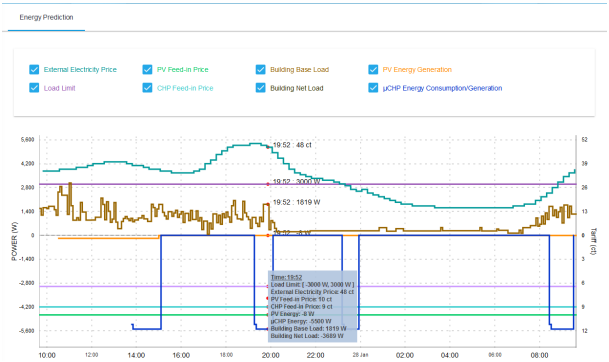


Figure 5.23: BOS UI for the resident: energy prediction of the building

- **Device operation**: The device is directly controlled by a resident by sending a command to it.
- **Group operation**: The device is indirectly controlled by a resident by sending a command to a device group which includes the device.
- **Scene trigger**: The device is indirectly controlled by a resident by triggering a scene which includes the device.
- **Calendar event response**: The device is controlled by the building operating system in order to respond to a calendar event specified by the resident.
- **System optimization**: The device is controlled by the building operating system that has defined the working schedule for the device in order to achieve a global optimization of the building. This operation mode is only applicable to appliances with a degree of freedom. The building operating system cannot change the working schedules for devices with low/no a degree of freedom, e.g. televisions, lights etc. since these devices can only be controlled by residents. Re-schedulable appliances which can be controlled to a certain extent

by the building operating system can be classified into three categories according to the adjustable direction of their degree of freedom. There are devices that can be re-scheduled bidirectionally on the timeline, e.g. refrigerators, devices that can only be re-scheduled backward on the timeline, e.g. hot-water boilers, and devices that can only be re-scheduled forward on the timeline, e.g. washing machines.

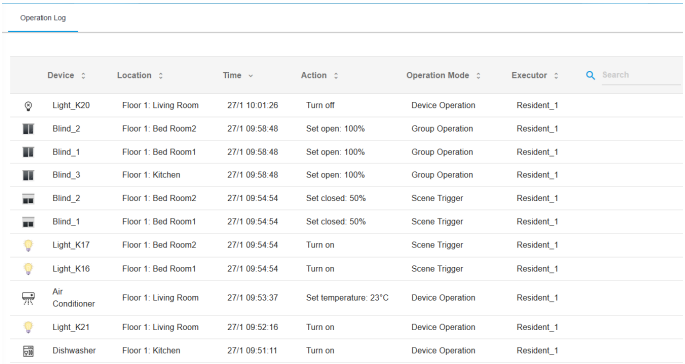


Figure 5.24: BOS UI for the resident: the device operation log

All the historical records can be displayed either in an ascending or descending order, according to any attribute of the records, and they can also be filtered by residents with corresponding keywords. By checking the operation log, residents are able to obtain the information about when, how and by whom their devices were controlled in the past.

The BOS UI is a web-based user interface which was implemented based on AngularJS technologies. Therefore, on the one hand, it can be accessed easily from any web browser by entering its URL address. On the other hand, the responsive layout enables the BOS UI to adapt to different screen sizes. Figure 5.25 shows several screenshots of a tablet which was used to access the BOS UI.

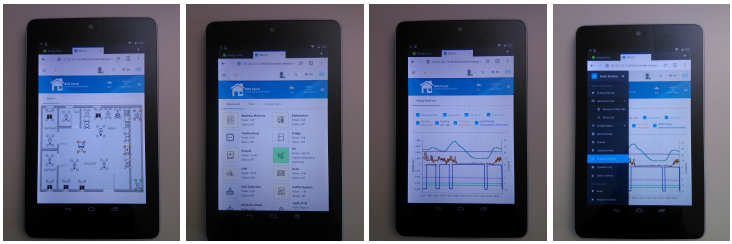


Figure 5.25: BOS UI on a tablet

In order to further improve usability, the BOS UI provides users with a number of global options which enhance user-friendliness. Firstly, because of the many functions covered by the BOS UI, a search service is available to enable users to quickly find the desired information by entering keywords into a search box. As for items that need to be accessed frequently, shortcuts can be generated for them on the top toolbar of the user interface. Secondly, because of the Angular Material library and the Fuse template, both layouts and visual themes of the BOS UI are configurable. Users are allowed to choose from available layouts and color schemes, thus providing them with personalized experience. Furthermore, the BOS UI gives users multilingual support so that they are able to switch between different languages on-the-fly without needing to refresh the page. This function is implemented with the help of angular-translate which is an AngularJS module. The languages that are currently supported by the BOS UI include English, German and Chinese. Extending the BOS UI to support other languages can be achieved by introducing their corresponding translation files to the system. These language translation files can be loaded asynchronously when users switch the display language for the BOS UI.

5.5 Conclusion and Discussion

Based on the design outlined in Chapter 4 of this thesis, the BOS UI prototype was developed, implemented and is being applied to a real smart home environment, namely, the ESHL at the KIT. The BOS UI is able to achieve the real-time communication with the OSH and the devices in the ESHL via the mechanisms of publication/subscription and the remote procedure call with the aid of a WAMP router. Furthermore, a number of RESTful Web Services are available in order to enable the BOS UI to access and operate the resources that are not constantly being updated in the ESHL. The BOS UI was developed on the basis of AngularJS Material. AngularJS Material is both a UI Component framework and a reference implementation of Google's Material Design Specification [3]. Benefiting from this technology, on the one hand, the BOS UI is capable of producing a rich and modern visual experience for users. On the other hand, functionalities of the BOS UI can be realized by means of using modules. A variety of modules have been created in order to provide the services needed by the UI framework and support various features of the BOS UI.

From the point of view of UI realization, the BOS UI in general consists of the navigation menu panel on the left-hand side, the toolbar at the top, and the content panel in the middle. Compared to the cockpit display (e.g. the FHEM Tablet UI), the tab-based display (e.g. the EMP), and any other forms of display that do not use menus (e.g. the Basic UI of openHAB, the OGEMA UI, the smartVISU, etc.), the system used by the BOS UI is able to display more content on the user interface in a logical and concise manner. Instead of displaying all of the content on one screen, the data that needs to be displayed on the BOS UI has been organized into different categories, whose titles can be found in the navigation menu on the left side of the screen. However, this navigation menu is also collapsable, which means, that it can either be locked open on the left-hand side of the user interface

or it can be hidden in order to expand the display area of the content for users. The information in the content panel is designed to be as intuitive as possible in order to make the user interface easy to use. The effectiveness, efficiency and degree of satisfaction arising from the way in which the BOS UI has been implemented, that, the usability of this user interface, will be evaluated in the next chapter.

Furthermore, as outlined in the design description in Section 4.5 of the previous chapter, the BOS UI was developed based on role-based access control. This can be seen by the way in which the menu options pertaining to different user roles have been integrated in the BOS UI into one navigation menu panel. To this end, the Login module of the BOS UI needs to ascertain the roles of users when they log into the system. The specific display options on the menu navigation panel for them will therefore depend on the respective roles which they hold.

Although the BOS UI has been applied to the ESHL, which is a real smart home environment at KIT, as indicated by the name, the ESHL is still a laboratory for research and demonstration. When it comes to realization of the design in this thesis in an actual household building or turning the prototype into a product, some challenges are still needed to be addressed. For instance, household buildings need to be equipped with intelligent appliances, which either have smart technologies built in or connect to extra smart plugs which allow energy consumption measurement and remote control to those conventional home appliances, so that an IoT environment could be created. However, intelligent appliances are currently much more expensive than conventional appliances. This impedes the transition from traditional homes to smart homes. Additionally, building operating systems are required to be installed to those household buildings since on the one hand, various household appliances from competing vendors need a unified platform to enable them to work together seamlessly. On the other hand, many features provided by the BOS UI, such as the realization of the optimization

goals in a building, need to be supported by the BOS UI's underlying building operating system. Besides this, since the user interface proposed in this thesis is designed in a generic way, a certain middleware or adapter should be prepared separately in order to implement the communication between the BOS UI and different building operating systems unless those building operating systems are compatible with the BOS UI. However, considering the current smart home market is highly fragmented and full of incompatible technologies, the implementation of middlewares for building operating systems with different standards requires also a great deal of operating expense.

6 Evaluation

This chapter evaluates the design, functionality and usability of the BOS UI by way of a theoretical and experimental analysis, respectively. It firstly evaluates the design of the BOS UI by checking to what extent the BOS UI meets the proposed criteria for a generic user interface for building operating systems. As for the functionality and usability, the BOS UI along with the original user interface of the ESHL, namely, the ESHL GUI, are evaluated at the same time by inviting test users to use both user interfaces and asking the test users to fill out questionnaires at the end. Part of the work in this chapter has been published in the Journal of Energy Informatics [106].

6.1 Evaluation of the Design

At the beginning of Chapter 4, a set of requirements or criteria, that are considered necessary for any generic user interface for building operating systems, were proposed. These requirements or criteria included remote reachability, responsiveness, configurability, role management, flexibility and generality. Specifically, as described in Section 4.1 of Chapter 4, for the discussion concerning generality of a generic user interface for building operating systems in this thesis, the use cases in Table 6.1 are used as a reference.

The aim of this section is to evaluate theoretically whether the BOS UI has met the required criteria. Figure 6.1 shows a summary of the evaluation results, and the following are explanations of the results:

Table 6.1: The use cases relating to a smart home

No.	Use Case	No.	Use Case
1	Basic Home Automation	10	Visualization of Historical Energy Costs
2	Advanced Home Automation	11	Visualization of Historical Energy Data
3	Possibilities to Specify Degrees of Freedom for Devices	12	Prediction of In-house Energy Use
4	Visualization of Building-level Energy Data	13	Support for System Configurations
5	Visualization of Device-level Energy Consumption	14	Provision of Value-added Services
6	Visualization of Device-level Energy Generation	15	Visualization of Historical Data for the Single Resident
7	Visualization of External Signals	16	Integration of Electric Vehicles
8	Role Based Access Control	17	Connection to a User Community
9	Floor Plan Based Device Organization	18	Support for Setting Building Optimization Goals

Remote Reachability	Responsiveness	Configurability	Role Management	Flexibility
✓	✓	✓	✓	✓
Generality				
use case 1	use case 2	use case 3	use case 4	use case 5	use case 6
✓	✓	✓	✓	✓	✓
use case 7	use case 8	use case 9	use case 10	use case 11	use case 12
✓	✓	✓	✓	✓	✓
use case 13	use case 14	use case 15	use case 16	use case 17	use case 18
☑	✓	✓	✓	✓	✓

✓: support ☑: partly support

Figure 6.1: Evaluation results for the design of the BOS UI

- Remote reachability. The BOS UI is a web-based single-page application, which is built for the web and can be accessed anywhere via any web browser by entering its URL address.
- Responsiveness. As discussed in Chapter 5, the BOS UI is developed on the basis of AngularJS Material library, which provides responsive layouts for different views (e.g. mobile, tablet, and desktop). Because

of the library, the BOS UI is able to adapt to different screen sizes, either by resizing or reorganizing its components on different views.

- Configurability. The BOS UI is configurable according to different aspects, including visual themes, layouts, languages and sizes and positions of the widgets, etc. One defect concerning the configurability of the current BOS UI is that most of the options customized by users cannot be saved for the next use once the user has logged out. This is something, which needs to be improved in the future by providing a configuration file for each of the users to record their personal settings instead of asking them to reconfigure the user interface every time after logging in.
- Role management. Role-based access control is one of the features supported by the BOS UI. To facilitate security administration and privacy protection, the BOS UI controls the users' access according to the roles held by the particular users and the permissions attached to these roles. For this purpose, three roles, namely, administrator, operator and resident are introduced in the BOS UI.
- Flexibility. The data models behind a user interface determine the flexibility of the user interface. In terms of the BOS UI, its data models are designed in a generic way, which means they do not exclusively apply to one specific building operating system in a particular household building. This contributes to the high flexibility of the BOS UI to extend to different building operating systems and cover various scenarios.
- Generality. As described in the definition of a generic user interface for building operating systems in Chapter 4, the generality of a user interface can be reflected in its support of a wide range of smart home related use cases (cf. Table 6.1). At this stage, all of the use cases in Table 6.1 can be covered by the current BOS UI except for the

use case dealing with the support of system configurations, which is currently only partly supported by the BOS UI.

The system configurations in this thesis refer to the configurations for the building operating system rather than for the user interface itself. So far, this use case is only supported by the BOS UI to some extent. On the one hand, the administrator is able to configure a building with respect to different aspects, such as location management, device deployment, etc. On the other hand, the BOS UI supports user management by allowing the operator to add/edit/remove residents and assign permissions to them to access devices in the system.

However, one limitation of the current BOS UI is that it can only manage the devices that have already been integrated into a building operating system. Discovering and adding new devices to the building operating system are not yet supported by the BOS UI on account of the heterogeneity of different building operating systems. The working mechanisms vary from one building operating system to another. Consequently, the configuration parameters for adding devices to their corresponding system may differ. For example, according to the introduction of the building operating systems in the Section 2.2 of Chapter 2, components including a proper device driver, an OX (Observer eXchanger) object and a CX (Controller eXchanger) object need to be loaded in order to add a device to the OSH. In order to add a device in openHAB, one needs to firstly import a so-called Thing to the system and then link Thing channels to predefined Items. Furthermore, for adding devices to EF-Pi, one needs to load the corresponding device drivers and device managers to the system. It is not challenging to create a custom user interface for adding devices to a specific building operating system. However, the BOS UI is designed as a generic user interface, which means that it is not tailored to any particular building operating system. Up until now, there is no such a

"one-size-fits-all" plan for adding devices or configuring various parameters for different types of building operating systems does not exist. For this reason, the BOS UI does not yet support this function.

Among all the data models of the BOS UI, the household device model (cf. Figure 4.6) is the innermost and therefore the most important model, since the other data models (e.g. scene, device group, location, etc.) are either made up of it or associated with it. Whether the household device model is generic or not determines whether the BOS UI is applicable to different building operating systems. To prove the generality of the household device model, the following are a few examples which illustrate the results of converting the proprietary device models from two building operating systems, namely, the OSH and the openHAB, into the generic household device model used by the BOS UI.

Example 1: an exemplary data representation of a washing machine in the ESHL used by the OSH

"-1609555631": {

"name": "Washing Machine",

"room": "kitchen",

"stateName": "Running",

"deviceDetails": {

"stateName": "Running",

"programName": "Delicates",

"phaseName": "Spin",

"remainingTime": "3",

"applianceTypeName": "Washing Machine"

}

"type": "W3985",

"class": 22020,

"uid": -1609555631

}

The washing machine in the OSH has more data pertaining to it than reflected in the aforementioned data representation. Other related information about a washing machine (e.g. power and degree of freedom, etc.) is stored separately in other, different data sets, as for other devices. In other words, the OSH does not provide a complete data model for any of the devices in the ESHL. Neither does the Energy Management Panel (EMP), which is the OSH's original user interface. The EMP accesses the energy data in the ESHL according to its understanding of the system. As a result, the design of the EMP is tightly coupled with the OSH, which makes it difficult to extend to other building operating systems. On the contrary, the BOS UI is implemented based on a number of generic data models. In order to apply the BOS UI to the OSH, the data from the OSH or the ESHL need to be converted into the format of the generic data models used by the BOS UI. For instance, the aforementioned information about the washing machine in the ESHL can be represented by the household device model of the BOS UI after conversion. Table 6.2 shows the result after the conversion. For the sake of brevity, part of the attributes, which have empty values, are not displayed in the table.

Example 2: an exemplary switch Item in the openHAB

Switch Bedroom_Light "Bedroom Light" <light> { mqtt="
> [mybroker:myhouse/bedroom/light:command:ON:1],
>[mybroker:myhouse/bedroom/light:command:OFF:0]"}

In openHAB, Items represent all properties and capabilities of the user's home automation, which are mainly used by user interfaces or the automation logic of an openHAB instance [22]. Items store different kinds of values which can be read or written, and on the other hand, they specify the way

Table 6.2: The result of converting the data about the washing machine in the ESHL from the OSH into the data model of the BOS UI

Uuid	-1609555631				
DeviceName	Washing Machine				
DeviceImage	washingmachine.png				
Device General Info	**infoName**		**infoValue**		
	room		kitchen		
	type		W3985		
	class		22020		
Device Channels	**channelName**	**channelInfo**			
		infoName	**infoValue**		**unit**
	Washing Machine	stateName	Off		
		programName	Delicates		
		phaseName	Spin		
		remainingTime	3		min
		power	442		w
Device Controllers	**controllerName**	**deviceActions**			
		name	**commands**	**widget**	**available**
	State Controller	turn on	**cmdString**: eshl.miele.v1.home-bus.start - 1609555631	Button	false
		turn off	**cmdString**: eshl.miele.v1.home-bus.stop - 1609555631	Button	true
DOFInfo	**allowedStartTime**		**requiredEndTime**		
	09:00, 07.02.2018		15:00, 07.02.2018		
Consumed Energy	electricity				
Generated Energy					

to connect with external physical devices. Devices involved in home automation can be represented by different types of Items (e.g. Color, Dimmer, Number, etc.) inside the openHAB world. The above example is a definition of a switch item which is used to describe a light in the bedroom. In like manner, the data can be easily converted into the generic data model of the

BOS UI. Table 6.3 shows the result of converting this switch Item into the household device model used by the BOS UI.

Table 6.3: The result of converting a switch Item in openHAB into the data model of the BOS UI

Uuid	Bedroom_Light				
DeviceName	Bedroom Light				
DeviceImage	light.png				
Device General Info	**infoName**		**infoValue**		
	room		bedroom		
Device Channels	**Channel Name**	**channelInfo**			
		infoName	**infoValue**	**unit**	
	Bedroom Light	state	Off		
		power	0	w	
Device Controllers	**Controller Name**	**deviceActions**			
		name	**commands**	**widget**	**available**
	State Controller	turn on/off	**state** 1 **cmdString** mybroker: myhouse /bedroom /light ON	Switch	true
			state 0 **cmdString** mybroker: myhouse /bedroom /light OFF		
DOFInfo	**allowedStartTime**		**requiredEndTime**		
Consumed Energy	electricity				
Generated Energy					

The aforementioned two examples show how to convert the exclusive data models of two building operating systems into the generic data model used by the BOS UI. Similarly, other building operating systems need to undergo the same adaptation so that they are able to utilize the BOS UI as their user interface. To this end, a middleware (cf. Figure 6.2) between the BOS UI and a building operating system is required. In order to complete the connection, at least the two components, namely, the Adapter and the BuildingConfig, need to be included in the middleware.

The Adapter is the component responsible for the data conversion. It fetches data from a building operating system and converts the data into the generic data models for the BOS UI. For example, the work of converting the data of two building operating systems in the aforementioned two examples into the data shown in Table 6.2 and Table 6.3 is done by the Adapter component. In order to do so, the Adapter needs to know the exact data structure of the information provided by the building operating system.

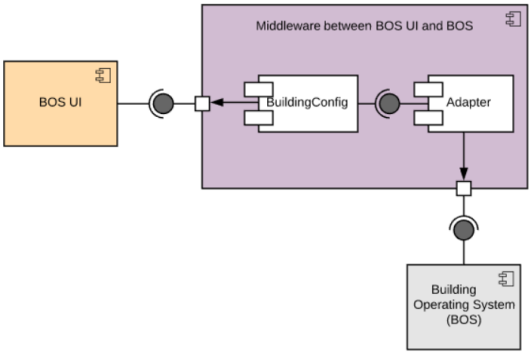


Figure 6.2: The component diagram of the middleware between the BOS UI and a building operating system

The BuildingConfig component, on the other hand, is responsible for providing all kinds of integrated services for the BOS UI to obtain information from the building or to communicate with the building operating system. For instance, it might provide a service to get all the devices that are consuming power in the building, or a service to get the detailed information for a specific device, and so on. In order to realize these integrated services, the BuildingConfig component usually needs to process and assemble some single energy values generated from sensors or other measuring equipment in a building in order to provide integrated services. This information is obtained not directly by communicating with the building operating system, but rather by invoking the interfaces provided by the Adapter. In so doing, the

BuildingConfig component does not need to deal with the building operating system, which allows the BuildingConfig component to be independent of building operating systems.

To sum up, because the BOS UI is equipped with this type of middleware between itself and the building operating system, it can be concluded that it is able to apply to different building operating systems. When it comes to the implementation on the market in the future, the middleware could be implemented by either building operating system providers or some other third parties which specifically provide services to implement a generic user interface for building operating systems. Building operating system providers can choose to design their own user interfaces by using their specific data models. Alternatively, they can also directly benefit from the design of the generic user interface in this thesis by using the generic data models to interact with their user interfaces. On the other hand, in the face of having many incompatible building operating systems on the market, there could be some third parties which provide generic user interfaces for customers to deal with different building operating systems installed in their household buildings in order to have a unified and holistic energy visualization and control at customers' houses.

Overall, according to the above analysis, it can be seen that, the BOS UI meets all the proposed requirements or criteria for a generic user interface for building operating systems, except for the fact, that the system configuration can not be fully supported yet. Functions such as adding new devices and configuring parameters for building operating systems still need to be done by the special configuration interface of each building operating system. The BOS UI is at this stage, mainly designed for the display and operation of the data in the building rather than the setup and configuration of the building operating system. The missing part of this function is relevant to the fourth initial research sub-question, namely, "What are the functional components that the generic user interface should cover?" The current solu-

tion provided by this thesis mainly focuses on the needs of residents rather than those of building operating systems. Therefore system configuration has not yet been included as a functional component into the generic user interface implemented in this thesis.

6.2 Evaluation of the Usability and Functionality

Usability is defined by ISO 9241-11 as the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [49]. Usability evaluation plays an important role in the overall user interface design process since usability provides an important contribution to user experience [72]. This section evaluates the usability of the BOS UI by testing it with end users which is the most fundamental usability evaluation method and is in some sense indispensable. It provides direct information about how people use the system and their exact problems with a specific interface [73]. Along with the usability, a range of important functionality aspects have also been evaluated using the same experiments.

6.2.1 Method of Experimentation

The evaluation of the usability and the functionality of the BOS UI was done by means of conducting experiments by inviting a number of test users to complete some pre-determined tasks and asking the test users to fill out questionnaires concerning the user interface. The detailed evaluation process is displayed by the flow chart in Figure 6.3. The first step was to design a test plan for the experiment, then to establish corresponding evaluation tasks and design questionnaires that needed to be completed by test users. The next step was to write the test users' instructions for the experiment in order to help the test users to deepen their understanding of the experimental pro-

cess and content. This step concluded the design phase of the experiment. The last step in the preparation of the test was to recruit a certain number of test users for the experiment. Before the actual test, pre-tests had to be conducted in order to eliminate potential errors in the code of the BOS UI and design flaws of the test plan. The actual test could get started as soon as no more problems were identified in the pre-tests. The last step of the experiment was to analyze the experimental results from which evaluation results of the usability and the functionality of the BOS UI could be derived.

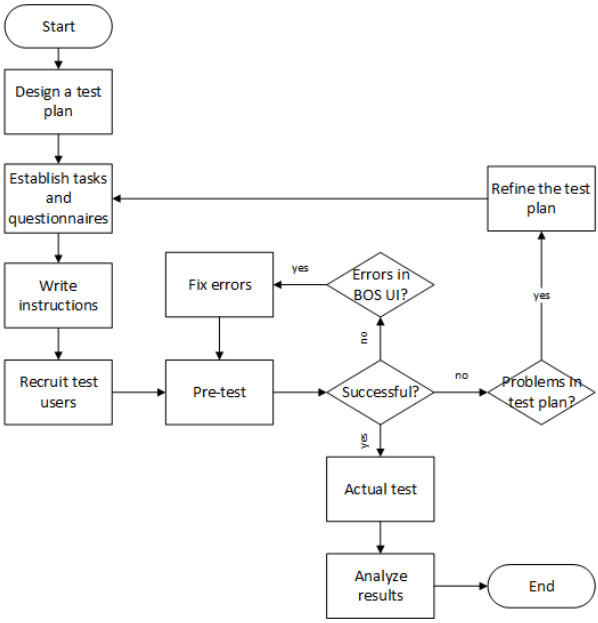


Figure 6.3: The experimental process of the usability evaluation of the BOS UI

Since the BOS UI was introduced as a replacement of a user interface specifically designed for the ESHL, named ESHL GUI, it made sense to evaluate the ESHL GUI along with the BOS UI, and then to provide a comparison between the two user interfaces. The ESHL GUI was further developed on

the basis of the Energy Management Panel (EMP) by integrating more diversified features into it. In addition to supporting the basic functionalities provided by the EMP, which were introduced in Chapter 3, the ESHL GUI is able to display more energy data, such as different sensor data, for users. Besides this, some other features which are more engineering-oriented, such as the visualization of parameters of different kinds of storage devices (e.g. hot water tank, cold water tank, battery, etc.) in the ESHL, and the visualization of various technical parameters about the power supply in the ESHL, were also integrated into the ESHL GUI in order to give users a more holistic understanding about the energy situation at the ESHL.

6.2.2 System Usability Scale

As mentioned above, the usability of the BOS UI and the ESHL GUI was evaluated in a user-based manner, namely, via a number of test users performing a set of pre-determined tasks, which are generally considered to yield the most reliable and valid estimate of an application's usability [61]. To establish a proper and effective way of measuring usability, a robust and reliable evaluation tool, named System Usability Scale (SUS), was used.

The SUS was invented by John Brooke in 1980s as a "quick and dirty" survey scale that allows the usability practitioner to easily estimate the usability of a given product or service. It has been tried and tested throughout 30 years of use and has proven a valuable and robust tool in helping assess the quality of a broad spectrum of user interfaces [42]. Besides this, as Brooke put it, the SUS is particularly relevant to compare two versions of an application that are based around different technologies [53].

The SUS is a Likert Scale which consists of the following ten statements. Each of them is given five response options from "strongly disagree" to "strongly agree" which represent different strengths of agreement.

1. I think that I would like to use this system frequently.

2. I found the system unnecessarily complex.

3. I thought the system was easy to use.

4. I think that I would need the support of a technical person to be able to use this system.

5. I found the various functions in this system were well integrated.

6. I thought there was too much inconsistency in this system.

7. I would imagine that most people would learn to use this system very quickly.

8. I found the system very cumbersome to use.

9. I felt very confident using the system.

10. I needed to learn a lot of things before I could get going with this system.

A final SUS score which represents a composite measure of the overall usability of the system being studied is yielded based on the answers from the respondents to the above questionnaire. To calculate the SUS score, the fact that the positive statements and negative statements in the questionnaire are alternately arranged needs to be considered. For statements 1, 3, 5, 7 and 9 the score contribution is the scale position minus 1. For statements 2, 4, 6, 8 and 10, the contribution is 5 minus the scale position, therefore the score contribution of each statement will range from 0-4. Multiplying the sum of the score contribution of each statement by 2.5 will yield an overall SUS score which has a range of 0-100. The greater the SUS score, the better the usability of the system being evaluated.

After analyzing more than 2300 surveys over the course of 206 studies, the mean SUS score for all surveys is 70.14 and the mean SUS score for Web user interfaces is 68 [42], which means 68 is around the 50th percen-

tile. In other words, a Web user interface's SUS score above 68 would be considered above the average and therefore, 68 can be taken as a minimal limit a Web user interface has to cross in order to be considered fairly usable. This is also mirrored in the acceptability estimate correlated to SUS scores. According to the analysis of nearly 1000 SUS surveys, an adjective rating scale which can help practitioners interpret individual SUS scores is highly correlated with SUS scores [41]. Figure 6.4 shows the corresponding relations between the SUS scores, the adjective ratings, the school grading scale and the acceptability ranges.

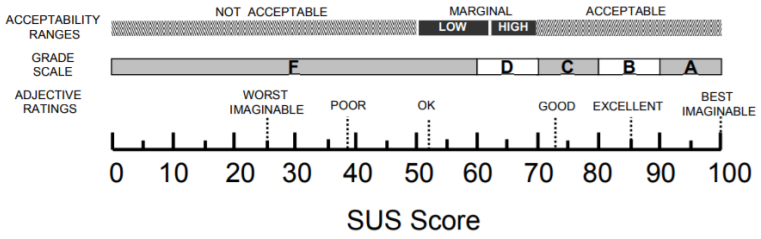


Figure 6.4: A comparison of the adjective ratings, acceptability scores, and school grading scales, in relation to the average SUS score [41]

The average SUS score for each adjective rating derived from the nearly 1000 SUS surveys is listed in Table 6.4, which can be used as a reference for judging how good the usability of the two user interfaces is in this evaluation according to their SUS scores.

6.2.3 Experiment Preparation

After this theoretical preparation, a series of experiments needed to be conducted in order to collect feedback from the test users on the two user interfaces. In this study, the experiments were carried out in the Karlsruhe Decision & Design Lab (KD2Lab)[1]. The KD2Lab is funded by the German

[1] https://www.kd2lab.kit.edu/english/index.php

Table 6.4: The statistical average values of SUS scores for adjective ratings according to nearly 1000 SUS surveys [41]

Adjective	**SUS Score**
Best Imaginable	90.9
Excellent	85.5
Good	71.4
OK	50.9
Poor	35.7
Awful	20.3
Worst Imaginable	12.5

Research Foundation. As one of the largest computer-based experimental laboratories world-wide, the KD2Lab can provide favorable conditions and psycho-physiological sensors for experiments, particularly for research on human behaviour and decision-making in economics and NeuroIS. For the experiments in this study, the KD2Lab offers 20 soundproofed and air-conditioned computer cubicles (cf. Figure 6.5). Every computer in the cubicles is installed with a screencasting software, which is used to capture and synchronize the screen during the experiment and output a video file for the purpose of analysis after the experiment. In addition to this, a cockpit cubicle is available for the experimenter to observe the test users' computer screens during the experiment. What is more, a microphone can also be used by the experimenter for announcing necessary information to the test users.

As for the participants of the experiments, the KD2Lab provides the experimenters with a participants pool which has more than 2800 registered users. With the help of a KD2Lab experimental portal, experimenters can invite any number of participants for their experiments by sending invitation emails to the users in the pool. Before starting to send the invitation emails, the KD2Lab experimental portal allows experimenters to filter the

Figure 6.5: The KD2Lab

users in the pool with some keywords, e.g. gender, language, degree, course of studies, etc., so that only the eligible users will receive the invitations.

In this study, no particular restrictions were placed on the participants except for language. The BOS UI supports switching between three languages, namely, English, German and Chinese, whereas, the ESHL GUI only offers German. As a result, the invitation emails were only sent to users who were able to speak both English and German.

The number of participants needed for a usability test was one of the most hotly debated issues in the field [38]. According to Nielson's article [89], testing with 5 people will find almost as many usability problems as the problems that would be found using many more test participants. However, for quantitative studies where statistics instead of insights are the aim, at least 20 participants are needed in order to get statistically significant numbers. According to a recent analysis of an internal SUS survey from SAP [23], 30 participants are needed to get a fairly accurate quantitative assessment of the overall quality of the system being studied. In order to ensure the reliability of the experimental results, around 10 participants were invited to attend the pretests. For the actual test, 42 participants were invited, which can be considered a large enough sample size to derive statistically stable insights.

In the experiment for this study, the objects to be evaluated were the two user interfaces: the BOS UI and the ESHL GUI. The participants of the experiment, namely, the test users, were asked to use first the one and then the other user interface for performing a number of pre-determined tasks. After that, they were asked to provide feedback for both user interfaces. To this end, a special evaluation website which integrates all the evaluation tasks and the questionnaires was developed for the participants. The organization structure of the evaluation website is illustrated in Figure 6.6.

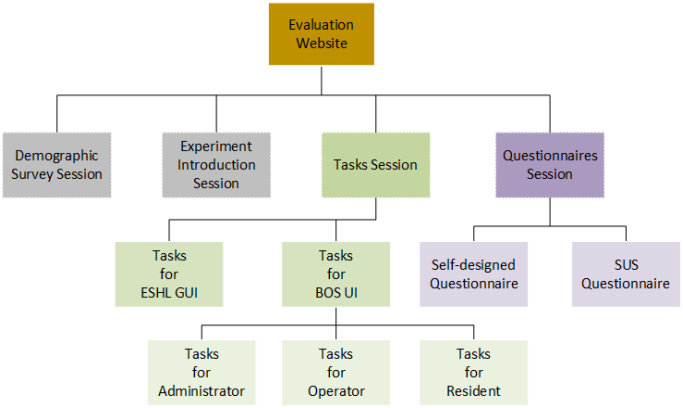


Figure 6.6: The organization structure of the evaluation website

6.2.4 Results of the Demographic Survey

After logging into the evaluation website, the first thing that the test users need to do was to fill out a demographic survey. The purpose of this section was to collect background information (e.g. age, major subject, degree, etc.) of the test users, the level of their knowledge about smart home technology and their familiarity with user interfaces for smart homes, if they had ever used one before. The complete background information of the test users which was collected from the demographic survey is listed in Appendix A.

In terms of knowledge about smart home technology, there were five levels available for participants to choose from, namely, no knowledge, basic knowledge, good knowledge, advanced knowledge and expert knowledge. In the experiment of this study, the ages of the 42 participants were between 18 and 40 years. The distribution of the number of participants with different knowledge levels about smart home technologies is shown in Table 6.5. Among the 42 test users, four had used smart home related user interfaces before. The description about the user interfaces and the participants' comments can be found in Table 6.6.

Table 6.5: The distribution of different knowledge levels about smart home technologies

Knowledge level	**Number of participants**
No knowledge	5 (11.9%)
Basic knowledge	28 (66.7%)
Good knowledge	7 (16.7%)
Advanced knowledge	2 (4.8%)
Expert knowledge	0 (0%)

Table 6.6: Smart home user interfaces that had been used by the participants before the test and their comments on the user interfaces

The user interface description	**Comment**
A user interface that can show all the states of the apartment.	At the beginning I was satisfied, but the more I used the interface, the more I felt that it missed some advanced features, e.g. time scheduled / event-based tasks.
It is an application to control the light in our home.	The application is really slow and not working sufficiently.
The sonos sound system	I am very happy with it.
Interfaces for heat regulation and air/ventilation system	N.A.

6.2.5 Main Part of the Survey

After the demographic survey, the evaluation website showed the test users some instructive information about the experiment and gave a brief introduction of the two user interfaces under evaluation. From the experimental introduction, the test users were able to obtain a general impression of what they needed to do during the experiment and how.

The test users were then asked to start the tasks for the BOS UI and the ESHL GUI. Since there are three roles in the BOS UI, the tasks for the BOS UI were organized into three parts according to the responsibilities of the different roles. The test users were given different roles to complete corresponding tasks on the BOS UI. In total, there were 22 tasks for the BOS UI which are listed in Appendix B.1. The ESHL GUI, on the other hand, does not support multiple roles, therefore only 8 tasks were designed for this user interface. These tasks can be found in Appendix B.2. Before the actual test, it could be seen from the results collected from the pre-tests that most of the test users only had a basic knowledge of smart home technologies and that they basically did not have any experience of using a user interface for a building operating system. In order to prevent the test users in the actual test from having a prejudice against any one of the two user interfaces, the evaluation website was programmed to display the tasks for the two user interfaces in different orders. More specifically, according to the username, with which the test users logged themselves into the evaluation website, half of the test users started with the tasks for the BOS UI and the other half started with the tasks for the ESHL GUI, so as to balance potential bias.

After the completion of all tasks, the test users were expected to be familiar with the two user interfaces. The last task which they needed to perform, was to fill out two different questionnaires. The purpose of the first questionnaire was to get the test users' overall impression of the BOS UI and the ESHL GUI, ask the test users to provide their views on the functionalities of

the two user interfaces, and then to make some comparisons between them. Consequently, the questionnaire includes a number of statements with different options and some questions about the two user interfaces. The second questionnaire consists of the aforementioned 10 standard SUS statements. The aim of this questionnaire was to evaluate the usability of the BOS UI and the ESHL GUI, respectively.

6.2.6 Evaluation Results of the Functionality

42 participants took part in the actual test in the KD2Lab. After processing and calculating the data collected from the experiment, the statistical results of the statements concerning the BOS UI in the first questionnaire are illustrated in Figure 6.7. There are six statements which exclusively deal with the BOS UI in the first questionnaire. The statements are listed as follows:

1. The BOS UI provided me with enough information and functionalities that I would need in my daily life based on my experience so far.

2. The BOS UI gave me a holistic view about the energy use in my building.

3. The BOS UI provided me with useful information which can help me to use appliances more reasonably in order to save money.

4. The BOS UI can help to achieve different optimization goals in the building, e.g. reducing energy costs or being environment-friendly, etc.

5. Having roles with different permissions to restrict system access is a crucial part of the user interface for smart buildings.

6. The way of showing devices on the floor plan of the building in the BOS UI is intuitive.

From the Figure 6.7, it can be seen that, except for one test user who remained neutral, all of the test users agreed (most of them strongly agreed)

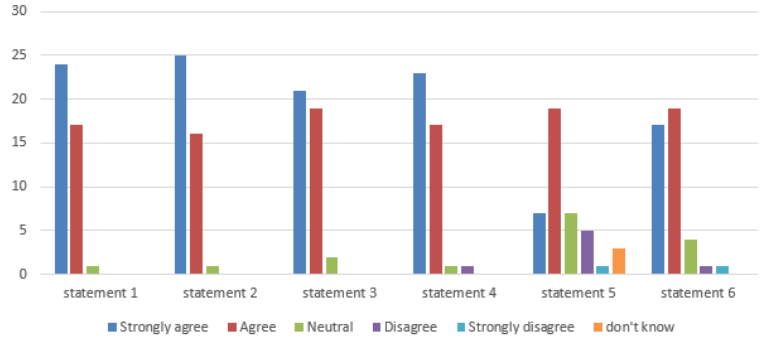


Figure 6.7: The statistical results of the six statements about BOS UI

that the BOS UI provided them with enough information and functionalities that they would need in their daily lives. 25 participants strongly agreed and 16 agreed that the BOS UI had given them a holistic view about the energy use in their building. Similarly, the third statement and the fourth statement were also strongly agreed to by half and more than half (23/42) of the participants, respectively. Compared to the first four statements, the test users' opinions about statement 5 and 6 were more diverse. Especially for the fifth statement, although more than half of the test users (26/42) agreed to this statement, there were still 6 test users who disagreed to it. According to the comments they left, they thought that having different roles might only be necessary in a big building in which more than one family lives, which is not the case in their current apartment, where everyone shares the same power. It is understandable that the test users made the decision based on their own circumstances. However, the BOS UI is designed not only for one particular situation, but for having the potential of covering a variety of use cases. In order to avoid unnecessary complexity and redundancy in simple situations, the BOS UI was designed on the basis of role-based access control (RBAC) in which permissions are associated with roles and users are made members of appropriate roles. A user, depending on different

situations, may have multiple roles. However, each user in the BOS UI only has one account, which means that he is able to access all the permissions owned by his roles by only logging in once. Because of this, the BOS UI is flexible and can be extended to apply to different situations.

As for the last statement, most of the test users (36/42) agreed that the way of showing devices on the floor plan of the building in the BOS UI is intuitive. While watching the recorded videos, it could be seen that two of the four test users who remained neutral and one of the two test users who disagreed with the statement, in fact did not turn to the menu item about displaying devices on the floor plan when they were using the BOS UI to perform the tasks. Instead, they tried to complete the related tasks by using other ways provided by the BOS UI. Since the three test users were not aware of this function, their choices for this statement should be considered as invalid. After removing the three invalid data sets from the total data set, the conclusion can be made that 92.3% of the test users believed that it is intuitive to show devices on the floor plan of the building after they had experienced this function. From this it can be concluded, that it is valuable to integrate this function into a user interface for building operating systems.

By replacing "BOS UI" with "ESHL GUI" in the statement 1 - 4, the test users also evaluated the ESHL GUI according to the same aspects as the BOS UI. The opinions expressed by the test users concerning the statements about the ESHL GUI were much more diverse. As shown in Figure 6.8, for any of these four statements, the number of test users who agreed with it only accounts for less than half of all of the participants. Most of the test users either remained neutral or disagreed with the statements.

In addition to the statements above, the test users in the first questionnaire were asked to choose the purpose for using the BOS UI. The statistical results are shown in Figure 6.9. Among the total of 42 test users, 35 of them believed that the BOS UI would be able to help them save money and 34 of them wanted to use the BOS UI to get a clear view of their energy use in

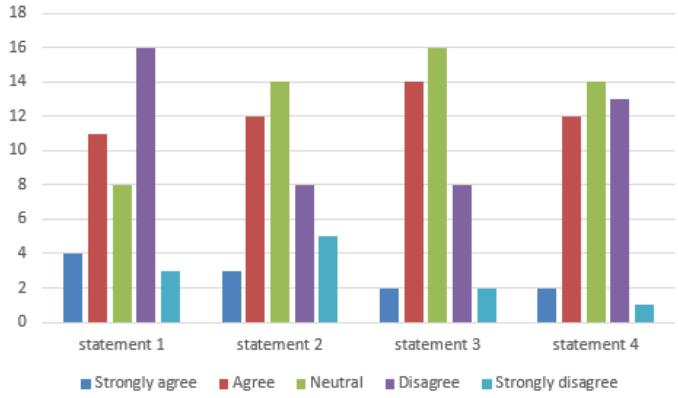


Figure 6.8: The statistical results of the first four statements about ESHL GUI

their building. There were 28 test users who would use the BOS UI for the purpose of convenience. In spite of functions like role-based access control, restricted permissions to residents, operation log, etc. provided by the BOS UI, fewer test users (12 to be exact) would use the BOS UI with the aim of improving security in their home. There were 5 test users who thought of other purposes for which they would like to use the BOS UI. The purposes which they expressed included: "schedule the things/machines in the building depending on my timetable", "control blinds and lights in the building while being on vacation", "control and to schedule/manage my devices", "if I have a large duplex house, it will definitely come in handy and allows you to check out the lights, windows, etc." and "I'd love to use it for convenience but it seems too heavy for now".

After getting familiar with both user interfaces by using them to perform a number of tasks, the test users were asked to make the following comparisons between the BOS UI and the ESHL GUI.

1. Compare BOS UI and ESHL GUI, which user interface offers more useful functionalities?

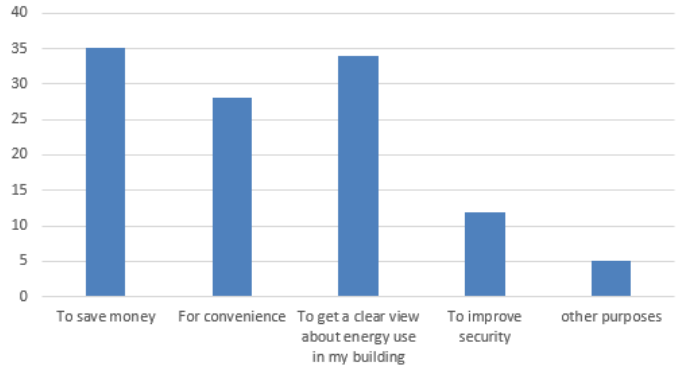


Figure 6.9: Purposes of using the BOS UI as stated by the test users

2. Compare BOS UI and ESHL GUI, which user interface is more intuitive to use?

3. Compare BOS UI and ESHL GUI, which user interface can help you save more money?

4. Compare BOS UI and ESHL GUI, which user interface provides more security features?

5. Compare BOS UI and ESHL GUI, which user interface has a bigger range of application?

6. Compare BOS UI and ESHL GUI, from which user interface did you get a more clear view about energy use in your building?

7. Considering the above reflections, which user interface do you like better?

The statistical results are illustrated in Figure 6.10, from which, one can see that, it would seem that, the vast majority of test users think that the BOS UI outperformed the ESHL GUI in many respects. When taking all

the aspects into consideration, 41 out of the total 42 participants considered that the BOS UI to be better than the ESHL GUI.

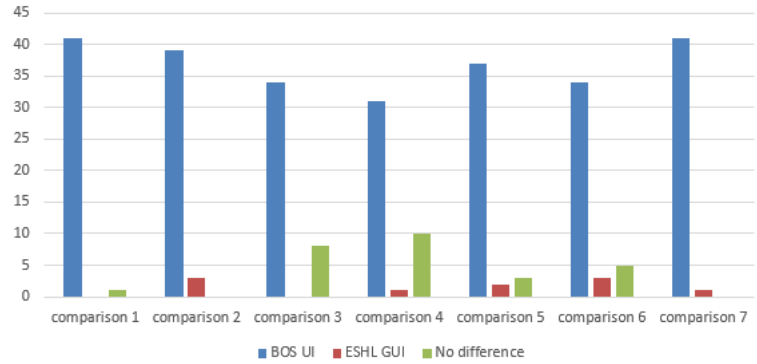


Figure 6.10: The statistical results of the comparisons between BOS UI and ESHL GUI

At the end of the first questionnaire, the test users were encouraged to leave their comments concerning the two user interfaces. The following are a few of the comments left by some of the test users. The complete list of comments concerning the BOS UI and the ESHL GUI, left by the test users in the test, can be found in Appendix C.

"BOS UI seems easy to use once you get used to it. It doesn't need much know-how to understand the interface. Whereas the ESHL GUI is not suitable for anybody like elder people who are not good with technology."

"BOS UI is extremely easy to use, so that it was quite fun. ESHL GUI on the other hand is annoyingly difficult. A lot of functionalities are missing."

"I really like the intuitive BOS UI for its very accessible interface with the different folders. Idiot-proof even for beginners. It was intuitive to find everything. The floor plan was great. I like the devices were coloured orange (consuming electricity) or green (producing electricity), giving a user a quick overview of what's going on.

I had some problems with ESHL GUI, it was quite clunky to work with. The devices weren't as comfortable accessible and the charts where to find what information were kind of confusing. More or better ways to group your devices or have an overview where the most power is used at this moment, maybe even on the floor plan, that would be great."

"The ESHL GUI is not very clear. You have to make a lot more clicks to get there. Also in terms of color, the BOS UI is much better designed, which improves clarity. However, in the Energy Overview tab, for example, I find the technical parameters unnecessary. The end user is certainly not interested in what voltage is currently available."

"I find ESHL GUI very unintuitive for use. For somebody who might be already familiar with this system it might be reasonable but I had a hard time finding specific features. BOS UI feels much smoother, however it would be great if BOS UI can keep track of the frequency of use of my devices so that all my devices can be sorted based on the frequency of their use."

It is suggested by the comments above, that the feedback of the test users concerning the ESHL GUI, is basically centered around complaining about its usability. The ESHL GUI is not considered to be intuitive and user friendly. On the contrary, the test users gave the BOS UI a positive evaluation in terms of usability, and in addition to that, provided some useful suggestions for the future, e.g. removing technical parameters that residents are not interested in and keeping track of user habits.

6.2.7 Evaluation Results of the Usability

In the second part of the questionnaire, the participants were asked to score the 10 statements of the SUS survey for the two user interfaces, respectively. After collecting answers from the participants and calculating, the mean value and the standard deviation of the 42 SUS scores for the BOS UI are

79.0 and 12.3, respectively. The results of the usability test for the BOS UI are positive. With the final SUS score of 79.0, which is 11 points more than the average SUS score for Web user interfaces, the corresponding adjective rating of the usability of the BOS UI is "good", according to Figure 6.4. The frequency distribution of the SUS scores of the BOS UI from the 42 test users is illustrated in Figure 6.11.

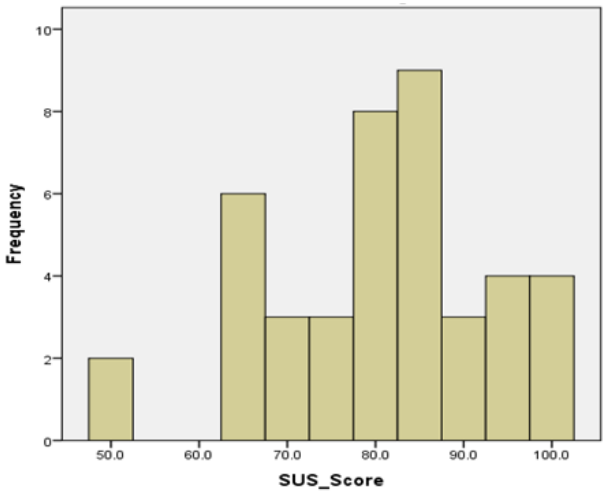


Figure 6.11: Histogram of the SUS scores of the BOS UI

By having the 42 discrete SUS scores, it was first assumed that this set of data was normally distributed, and then this assumption was checked by making use of statistical analysis. The Q-Q plot (quantile-quantile plot), which displays the observed values against normally distributed data, is one of the visual methods to check normality of sample data. Figure 6.12 shows the Normal Q-Q Plot of SUS scores of the BOS UI from the 42 test users, from which it can be seen that the points form a line that is roughly straight. The Q-Q plot in Figure 6.12 provides a visual judgement that the data set conforms to a normal distribution.

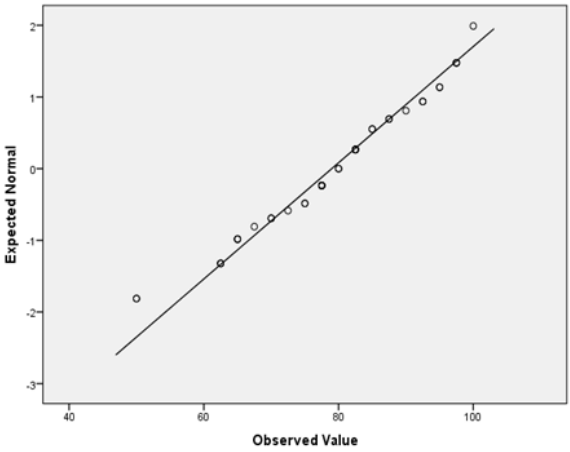


Figure 6.12: Normal Q-Q Plot of the SUS scores of the BOS UI

It is preferable that normality be assessed both visually and through normality tests, of which the Shapiro-Wilk test, provided by the SPSS software, is highly recommended [66]. Compared to the visual method, statistical tests for normality are more precise since actual probabilities are calculated. In this case, both the Kolmogorov-Smirnov (K-S) with Lilliefors correction and the Shapiro-Wilk tests were applied to test the normality of the SUS scores from the 42 test users. The results generated by the SPSS software are shown in Table 6.7. It is clear that both tests have a p-value greater than 0.05, which indicates normal distribution of the SUS scores.

Table 6.7: Tests of normality of the distribution of the SUS scores of the BOS UI

	Kolmogorov-Smirnova			Shapiro-Wilk		
	statistic	df[b]	p-value	statistic	df[b]	p-value
SUS score	.119	42	.149	.965	42	.214

a. Lilliefors Significance Correction b. Abbreviation: df, Degree of freedom

It can be seen from the above analysis that the SUS scores of the BOS UI from the 42 participants conform to a normal distribution. If the SUS score is presented by the random variable X, then: $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the mean of SUS scores for the BOS UI, which is 79.0, and σ represents the standard deviation of the distribution, which in this case is 12.3. With this information, it can be calculated, that the probability that the SUS score for the BOS UI is higher than 68 (which is the average SUS score of the Web user interfaces) is 81.4%. Based on the statistical average values of SUS scores for adjective ratings in Table 6.4, the probability, that users think the usability of the BOS UI is "good" or higher, is 73.2%. There are 29.9% of users who would consider the usability of the BOS UI to be "excellent" and "best imaginable".

Compared to the BOS UI, the usability of the ESHL GUI was graded less favorably. The statistical results show that the SUS score of the ESHL GUI is only 34.8, which is far less than the average SUS score, namely, 68. According to Figure 6.4, the adjective rating corresponding to this SUS score is in between "awful" and "poor". The frequency distribution of the SUS scores of the ESHL GUI from the 42 test users is shown in Figure 6.13, where it can be seen that only 5 test users (which takes up around 12% of the total) scored the usability of the ESHL GUI greater than 68.

When it comes to the test users' answer to the statements in the SUS survey, it would be better to divide them into two parts for statistics according to the parity of the number of the statements since the positive and negative statements in the SUS survey were alternately arranged. Figure 6.14 illustrates the results of the participants' answers to the odd numbered statements about the BOS UI and the ESHL GUI by means of stacked bars.

Figure 6.14 shows that 32 of the 42 test users agreed that they would like to use the BOS UI frequently. Only 3 test users disagreed with this. As for the ESHL GUI, most of the test users thought they would not like to use it frequently. The third statement was about ease of use. 39 test users, which

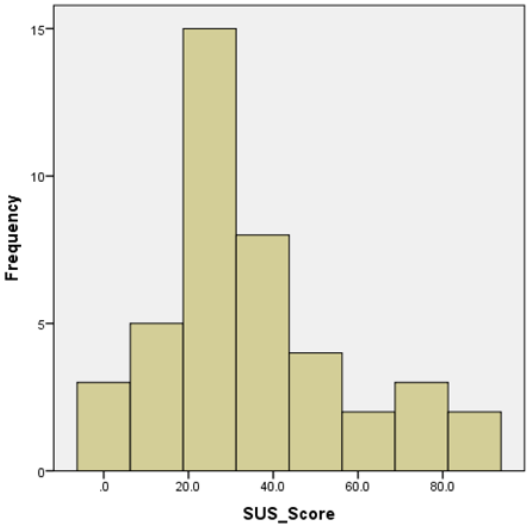


Figure 6.13: Histogram of the SUS scores of the ESHL GUI

accounts for 92.9% of the total, agreed that the BOS UI is easy to use, while for the ESHL GUI, the feedback for this question was quite the opposite. There are 38 test users who did not think that the ESHL GUI is easy to use. 95.2% of the test users found that the various functions in the BOS UI were well integrated and the same number of test users could imagine that most people would be able to learn how to use the user interface very quickly. However, only 5 test users and 9 test users, respectively, agreed on the two statements when applied to the ESHL GUI. Finally, the number of test users who felt very confident in using the BOS UI is 39, which stands in stark contrast to 5 users who felt this way about using the ESHL GUI.

The even numbered statements in the SUS survey are expressed in a negative way, therefore for these statements, the more the test users disagree with them, the better. The results of the test users' answers to these statements for the BOS UI and the ESHL GUI can be found in Figure 6.15. Although a

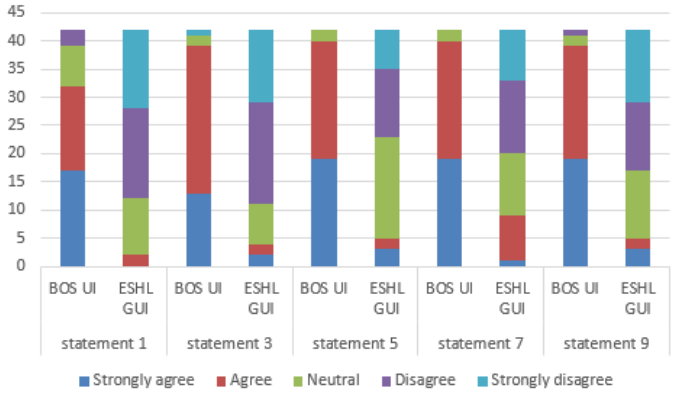


Figure 6.14: Stacked bar chart of the participants' answer to the odd numbered statements in the SUS questionnaire concerning the BOS UI and the ESHL GUI

large number of features are integrated in the BOS UI, 33 test users, which takes up 78.6% of the total, did not agree that it is unnecessarily complex. Compared to the BOS UI, the ESHL GUI does not provide many functionalities, nevertheless, more than half of the participants found it unnecessarily complex. There is only one test user who thought he/she would need the support of a technical person in order to be able to use the BOS UI. 13 test users, however, had this same feeling about the ESHL GUI. Another sharp contrast arises from the statement regarding inconsistency. The number of test users who thought that there was too much inconsistency in the BOS UI and the ESHL GUI were 1 and 19, respectively. Statement 8 uses the opposite way to express the same problem addressed in statement 3. The results indicate consistency of the answers to the two statements given by the test users. Nobody found the BOS UI very cumbersome to use, but 29 test users found the ESHL GUI cumbersome. In the end, 30 test users did not agree that they needed to learn a lot of things before they could get going with the

BOS UI, while more than half of the test users agreed with this statement when it was applied to the ESHL GUI.

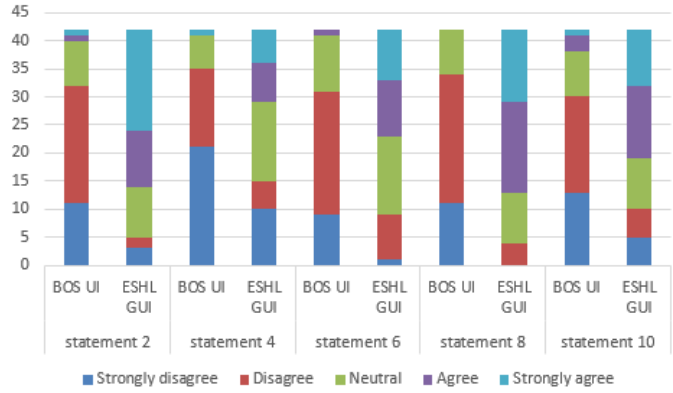


Figure 6.15: Stacked bar chart of the participants' answer to the even numbered statements in the SUS questionnaire about the BOS UI and the ESHL GUI

Figure 6.16 shows the SUS scores of the BOS UI and the ESHL GUI from the 42 test users after sorting them into a descending order, so as to reveal the significant difference in scores between the two user interfaces. Statistical analysis can further be done from the perspective of the smart home related knowledge that the test users had. As shown in Table 6.5, there were two test users who had advanced knowledge about smart home technologies. The SUS scores which they gave for the BOS UI are 97.5 and 90, respectively. The corresponding adjective rating that these scores match is "excellent". The ESHL GUI, on the other hand, got 30 and 17.5, respectively, from these two test users. The average SUS score of the BOS UI from the test users who had a good knowledge of smart home technologies is 78.9, which is almost the same as the overall average SUS score of the BOS UI. The corresponding SUS score of the ESHL GUI is 38.2, which is greater than its overall averages. Those, who knew nothing about smart

home technologies, scored the BOS UI and the ESHL GUI with averages of 69.5 and 28.5, respectively, which are much lower than their corresponding overall average. This is, however, not surprisingly since the test users lacked the necessary background knowledge. Most test users had a basic knowledge about smart homes. These test users also have relatively high scores on the BOS UI and the ESHL GUI. The averages of the two user interfaces for these test users are 81.0 and 37.1, respectively. In addition, there were 4 test users who had experience of using some kind of user interface for smart home before the experiment. The average SUS score they gave to BOS UI is 88.8. The adjective rating that matches the score is "excellent". However, the average score that these test users gave to the ESHL GUI is only 18.75.

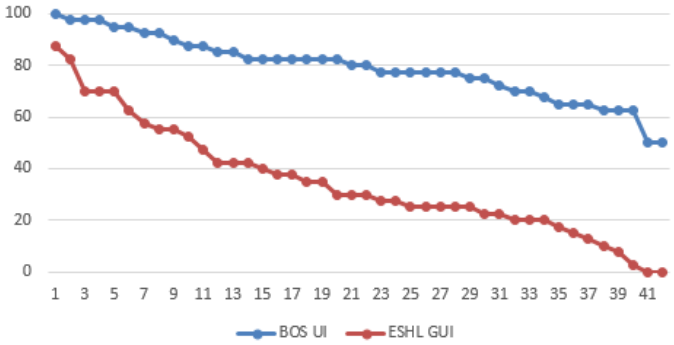


Figure 6.16: The SUS scores of the BOS UI and the ESHL GUI from the 42 participants

Finally, the test users' completion rate of the tasks related to the BOS UI and the ESHL GUI is also different. Appendix B lists all of the tasks designed for both user interfaces. As for the BOS UI, there were 4 tasks for the role of administrator. All of the test users completed these tasks successfully during the experiment, which leads to a completion rate of 100%. There were 6 tasks for the role of the operator. Except for the fact that each of 4

test users failed one task, the other test users completed every task successfully, which leads to a completion rate of 98.4%. 12 tasks were prepared for the role of resident. Many test users did not successfully complete all of the tasks, especially the first two, which were only partially completed by these test users. The reason was, that their answers either missed some information, or that the test users could not find the right information in the proper place of the BOS UI. The completion rate of the 12 tasks related to the resident is 88.7%. As for the ESHL GUI, test users needed to work on 8 tasks in total. The tasks, that could not be completed successfully, mainly concerned the first task and the third task. Overall, the completion rate of all the tasks for the ESHL GUI is 68.8%. Many test users had problems finding an effective solution for the tasks with the ESHL GUI due to the inferior usability of the user interface. The detailed information concerning achievements of these tasks related to both user interfaces can be found in Appendix D.

6.2.8 Discussion

According to the test users' feedback in the two questionnaires, it can be concluded that the BOS UI has made great improvements both in terms of functionality and usability relative to the ESHL GUI. In addition to the lack of many functions needed by the test users in their daily life and the shortcomings in the design of the ESHL GUI, there are several non-technical reasons that could have caused the test users to make more favorable statements about the BOS UI than about the ESHL GUI.

Firstly, the ESHL GUI was specifically designed for the Energy Smart Home Lab (ESHL) at the KIT. Its major tasks are to provide transparent information about energy consumption and generation in the ESHL for the residents who are already familiar with it and to discover degrees of freedom of the devices in the ESHL. Therefore the original intention of the

ESHL GUI was not to be used as a general user interface which focuses on serving uncertain users who know nothing about it in an efficient and a user friendly way. This makes its learning curve very steep for beginners. All test users in the experiment had no prior knowledge about the ESHL GUI. As a result, most of them experienced many difficulties finding out how the user interface worked, especially when they tried to work things out sitting in a booth of the laboratory, where they were generally not very relaxed and could easily become impatient.

Secondly, the ESHL GUI was further developed on the basis of the previous Energy Management Panel (EMP) in the ESHL by integrating many new engineering-related features. For instance, in addition to the common true power information about the devices, the ESHL GUI also displays reactive power which is measured in the unit of Volt-Amps-Reactive (VAR) for some devices as well as for the entire energy use in the building. Only professionals, and not ordinary residents can understand the term of reactive power. Besides this, some other information provided by ESHL GUI, such as power network parameters (e.g. voltage, frequency, current, etc.), the identifier and residual electricity of the Bluetooth Low Energy / Smart (BLE) beacons owned by the people working for the ESHL and the temperature of the both warm and cold water storages, etc. did not seem to be of much interest to the test users. Although this engineering-focused information was not involved in the tasks that were asked to be performed by the participants, this information could still distract the test users more or less.

Thirdly, the organization form of the content in the ESHL GUI was unclear and even overwhelming for some of the test users. By integrating too much information into one page and repeating some information on different pages, the ESHL GUI made the test users feel confused while they tried to find out the organizational logic of the content of the ESHL GUI. What made the participants even more frustrated is the lack of text descriptions or labels for the icons representing different functions in the ESHL GUI. Many

test users left their comments to complain that the ESHL GUI was either too abstract or too complex to understand and their suggestion was to add explanations to the icons and group the functions according to their features rather than showing them all in one page without any clues. By doing so, the user interface might become more tidy, intuitive and user friendly.

Although when compared to the ESHL GUI, the BOS UI has been greatly improved in many respects, the SUS score given by the test users indicates that its usability is good but not yet excellent, which means that there is still room for improvement. Many test users left valuable feedback on this. For instance, some of them suggested adding a submit button or popping up timely feedback for users after some operations have been performed, so that users may know whether the operations they performed worked or not. Also, the BOS UI at this stage, does not respond very fast to certain operations since a large amount of live data related to device states needs to be updated frequently in the back-end in order to ensure that the BOS UI can reflect the power use in the ESHL in real time. The delay in the response is a factor that can affect user satisfaction. There are a few of test users who commented that the BOS UI, with its many options on the left menu panel, was complicated to use. The suggestions they gave include hiding infrequently used options and regrouping options in the menu panel to make them tidier. Apart from the evaluation results of the two user interfaces, the biggest achievement of conducting this experiment was the feedback received from the test users. No matter whether positive or negative, they provided valuable information that helped not only to uncover defects in the current system but also formulated good suggestions for enhancing the performance of the BOS UI in the future.

7 Conclusion and Outlook

After designing, implementing and evaluating a generic user interface for building operating systems, this chapter concludes the work that has been done in the previous chapters and analyzes the achievements of this thesis. Subsequently, it provides an outlook for future systems and gives suggestions for the further development of the work presented in this thesis for the future.

7.1 Conclusion and Contribution

At the beginning of the thesis, a major research question was raised: *How should a generic user interface be designed, so that it can deal with different kinds of building operating systems while ensuring good usability?* In order to provide a solution for this research question, in the ensuing chapters of the thesis, the concept of a generic user interface for building operating systems was proposed and a complete study on the topic, including design, implementation and evaluation was presented. This is also the major contribution of this thesis. The major research question was divided into multiple sub-questions, as considered from different perspectives. These sub-questions were concretized and answered one by one in this thesis.

The first sub-question was "*What is the definition of a generic user interface for building operating systems?*" This question was dealt with in Section 4.1, where a generic user interface was defined as having to meet the criteria of remote reachability, responsiveness, configurability, role ma-

nagement, flexibility, and generality. The reasons why these criteria are important were also explained in this section. The proposed definition for a generic user interface not only identifies guidelines for its design, which was outlined in the following part of the thesis, but also provides criteria for the evaluation of the prototype of a generic user interface, implemented on the basis of this design.

The second sub-question was "*What does the architecture have to look like for such a generic user interface?*" This question was answered in Section 4.3, where an architecture of generic user interfaces for building operating systems in smart homes was proposed. The architecture shows that the essential part that makes a user interface generic and extendible, is a collection of generic data models, which are appropriate abstractions for objects that are needed by the generic user interface. In order to make a generic user interface applicable to existing building operating systems, an adapter component, which is responsible for the data conversion between the user interface and these building operating systems, is indispensable. Furthermore, the architecture also indicates that many favorable services, which can be used to reach some external stakeholders, such as communities, social networks and utility companies, should be provided by a generic user interface for building operating systems.

The third sub-question is "*What are data models for such a generic user interface?*" This question was answered in Section 4.6. As described previously, the central part that enables a generic user interface to uncouple itself from its underlying building operating system is a series of generic data models. For this reason, this thesis has proposed a number of data models, which are generic and flexible enough to be able to deal with heterogeneous building operating systems of different household buildings. Besides this, these data models enable a generic user interface to provide multiple possibilities not only for users to manage their building in a smart way but

also for building operating systems to exploit potentials of load shifting in household buildings.

The fourth sub-question is "*What are the functional components that the generic user interface should cover?*" According to a wide range of smart home related use cases presented in Section 3.2, a number of functional components that should be supported by the a generic user interface for building operating systems were listed and explained in Section 4.7, which, therefore, provides answers to this research question. The proposed functional components can be classified into two categories, namely, the functional components which are related to user roles and the functional components which are related to the whole UI framework. Three roles, namely, the administrator, the operator and the resident, were proposed in this thesis in order to facilitate the separation of responsibilities and improve security and privacy protection. Nevertheless, as discussed in Section 6.1 of Chapter 6, due to certain challenges, the functional component of system configuration is missing in the current implementation of the user interface in this thesis. So far, the solution provided by this thesis mainly lays stress on the needs of residents rather than those of building operating systems.

The last sub-question is "*How can such a generic user interface be made configurable and extendible?*" This question was answered in two ways. On the one hand, the generic data models designed for such a generic user interface for building operating systems, as well as the role-based access control, enable the user interface to extend to a number of household buildings and adapt to different situations. For instance, the building model (cf. Figure 4.10) provides the basis for the implementation of the building configuration on the generic user interface. At the same time, many more configurable attributes, such as layouts, visual themes, languages, etc. have been added to the prototype of such a generic user interface, namely, the BOS UI, to allow users to customize the user interface according to their preferences.

In order to facilitate the evaluation of the design, the BOS UI was implemented and applied to a real smart home environment, namely, the ESHL at KIT. The functionalities that have been implemented in the BOS UI were described in detail in Chapter 5.

At the end of the thesis, the design, functionality and usability of the BOS UI were evaluated by combining theoretical analysis and experiments. The following results can be derived from the evaluation.

- In terms of the design, the BOS UI meets the proposed criteria for a generic user interface for building operating systems, except for the fact, that the system configuration can not be fully supported yet.
- In terms of the functionality, almost all (more than 90%) of the test users in the test agree that: (1) the BOS UI can provide them with enough information and functionalities that they would need in their daily lives; (2) the BOS UI can give them a holistic visualization about the energy use in their building; (3) the BOS UI can provide them with useful information which can help them to use their appliances more reasonably in order to save money; and (4) the BOS UI can help them to achieve different optimization goals in their building. The statistical results also show that most of the test users prefer to use the BOS UI for the purpose of saving money, getting a clear view of the energy use in their building, and to increase convenience.
- In terms of the usability, a robust and reliable evaluation tool named System Usability Scale (SUS) was used. The final SUS score of the BOS UI in the experiment indicates that the usability of the BOS UI is good. More specifically, the vast majority of the test users believed that (1) they would like to use the BOS UI frequently (76.2%), (2) the various functions in the BOS UI are well integrated (95.2%), (3) the BOS UI is easy to use (92.9%). (4) they felt that most people would

learn to use the BOS UI very quickly (95.2%), and (5) they felt very confident about using the BOS UI (92.9%).

The ESHL GUI, which is the original user interface of the ESHL at the KIT, was also evaluated in the experiment, along with the BOS UI. It can be seen from the evaluation results that the BOS UI makes great improvements in many aspects when compared to the ESHL GUI. The reasons which could cause the test users to make unfavorable statements about the ESHL GUI were analyzed at the end of Section 6.2. Furthermore, many test users gave valuable comments concerning the two user interfaces, which on the one hand, affirm the advantages of the user interfaces, on the other hand, point out the shortcomings of the user interfaces that need to be improved in the future.

7.2 Outlook and Future Work

Although the BOS UI received much positive feedback from the test users in the evaluation experiment, there are many potential extensions to the current work that may help to strengthen the functionality of the user interface so as to expand its scope of application. As described previously, the BOS UI thus far does not support the configuration of its underlying building operating system due to the heterogeneity of different building operating systems. To this end, an additional interface is still needed in order to complete the system configuration. To address this challenge, potential future research may focus on collecting and analyzing configuration requirements of various building operating systems and designing flexible components for the generic user interface which can be customized to meet different system configurations.

As for the configuration of the user interface itself, the functionality of individual customization of the BOS UI can be further strengthened by providing a configuration file for each user so as to save their configured prefe-

rences. Besides this, according to the feedback received from the test users in the evaluation experiment, it would be beneficial for users to have the freedom to configure the user interface to show or hide some of the menu items according to the frequency at which some of the functions provided by the user interface are used. Giving users as much freedom as possible to customize their own user interface is another area in which it is worth investing in the future.

Some of the functionalities provided by the BOS UI at this point in time can also be further extended in the future. The floor plans of a building are one example: Presently, they are images that are uploaded to the BOS UI by the administrator. After uploading a floor plan, device icons need to be placed in their proper positions on the floor plan according to the actual location of the devices in a building. To this end, the administrator needs to, first of all, prepare floor plan images for a building and then manually place the device icons onto the floor plan images. The potential future work that might enhance this function could be to integrate a floor plan designer into the BOS UI. A floor plan designer would enable the administrator to create logical layouts for any floor in the building. By establishing an association between the actual location of devices in a building and the logical locations designed on a floor plan via the floor plan designer, device icons could automatically be loaded to the corresponding locations of a floor plan. However, even if this could be achieved, device icons on a floor plan would inevitably still need to be slightly adjusted, since different devices are usually at different positions of a building, even if they are in the same room. Finding a solution which will not only maximize convenience but also minimize effort for users to configure their building is also part of the suggested future work.

Furthermore, the BOS UI currently focuses on the aspects of comfort and energy relating to smart homes. Future research might be done toward other aspects pertaining to smart homes, such as healthcare and security. Another interesting aspect for future designs could be the integration of user interfa-

ces for energy into audio or audio-visual interfaces. The increasing popularity of smart speakers, such as Amazon Alexa and Google Assistant, indicate that the trend towards voice-controlled user interfaces would become mainstream in the future. Besides this, according to the feedback of the test users in the evaluation experiment, it could be of advantage if some features, such as multi-room digital music systems and security alarms could be integrated into the BOS UI in the future. Additionally, some other features such as privacy-enhancing energy visualization [43], which implements mechanisms to avoid surveillance of fellow residents, could also help to enrich and strengthen the functionality of the user interface.

In conclusion, with smart home technologies becoming increasingly popular, it can be expected that residents of smart homes will, in the future, also gradually raise their expectations and requirements for the user interfaces of building operating systems. Because of this, it can be foreseen that, in addition to the work mentioned above, more extensions and improvements will need to be made continuously in the future in order to ensure that the user interface keeps pace with the times.

A Background Information of the Test Users in the Evaluation Experiments

Table A.1: Background information of the test users in the evaluation experiments

No.	Gender	Age	Education	Major Subject	Knowledge about smart home technologies	Experience about UI for smart home	The description of the UI that was used
1	Male	30	Bachelor	Mechanical Engineering	Nothing	No	-
2	Male	26	Master	Economics	Basic	No	-
3	Male	25	Master	Industrial engineering	Basic	No	-
4	Male	26	Bachelor	Industrial engineering	Basic	No	-
5	Female	22	Master	Industrial engineering	Nothing	No	-
6	Female	20	Master	Industrial engineering	Nothing	No	-
7	Male	22	Bachelor	Technology	Basic	No	-
8	Male	24	Bachelor	Education	Nothing	No	-
9	Female	30	Bachelor	German studies	Basic	Yes	Interfaces for heat regulation and air/ventilation system
10	Male	40	Lectureship	Chemistry	Nothing	No	-

Continued on next page

Table A.1 – continued from previous page

No.	Gender	Age	Education	Major Subject	Knowledge about smart home technologies	Experience about UI for smart home	UI description
11	Male	19	Bachelor	Electrical and information engineering	Basic	No	-
12	Male	20	Bachelor	Physics	Basic	No	-
13	Male	19	Bachelor	Electrical Engineering	Basic	No	-
14	Male	21	Bachelor	Mechatronics	Basic	No	-
15	Male	26	Bachelor	Computer science	Advanced	Yes	I can't remember the name, but the interface showed all the status of the apartment. At the beginning I was satisfied, but the more I used the interface, the more I felt that it misses some advanced feature, e.g. time scheduled/event-based tasks.
16	Female	27	Bachelor	Information technology	Basic	No	-
17	Male	22	Bachelor	Electrical engineering	Basic	No	-
18	Female	22	Bachelor	Biology	Basic	No	-
19	Male	21	Bachelor	Mechanical engineering	Basic	No	-
20	Male	24	Master	Economics engineering	Good	No	-

Continued on next page

Table A.1 – continued from previous page

No.	Gender	Age	Education	Major Subject	Knowledge about smart home technologies	Experience about UI for smart home	UI description
21	Female	22	Bachelor	Chemical engineering	Basic	No	-
22	Male	29	Master	Industrial engineering	Basic	No	-
23	Male	19	High-school diploma	Engineering	Basic	Yes	I have used the sonos sound system and i am very happy with it.
24	Female	23	Master	Food chemistry	Basic	No	-
25	Male	22	Bachelor	Electrical engineering	Basic	No	-
26	Male	22	Bachelor	Computer science	Advanced	No	-
27	Female	23	High-school diploma	Computer science	Good	No	-
28	Male	37	Bachelor	History	Good	No	-
29	Male	27	Bachelor	Computer science	Basic	No	-
30	Male	29	Bachelor	Economical engineering	Good	No	-
31	Male	25	Master	Mathematical economics	Basic	No	-
32	Male	22	Bachelor	Electrical engineering & information technology	Good	No	-
33	Male	26	Master	Industrial engineering	Basic	No	-
34	Male	20	High-school diploma	Mechanical engineering	Basic	No	-

Continued on next page

Table A.1 – continued from previous page

No.	Gender	Age	Education	Major Subject	Knowledge about smart home technologies	Experience about UI for smart home	UI description
35	Male	20	Bachelor	Chemical engineering	Basic	No	-
36	Female	26	Bachelor	Mechanical engineering	Basic	No	-
37	Male	26	Bachelor	Electronic engineering	Basic	No	-
38	Male	23	Bachelor	Industrial engineering	Good	No	-
39	Female	21	Bachelor	Electrical engineering	Basic	No	-
40	Male	21	Bachelor	Mechanical engineering	Good	Yes	It is an app to control the light in our home. The app is really slow and not working sufficiently.
41	Male	23	Master	Information economy	Basic	No	-
42	Female	19	Bachelor	Computer science	Basic	No	-

B Tasks for the BOS UI and the ESHL GUI in the Evaluation Experiments

B.1 Tasks for the BOS UI

B.1.1 Tasks for the Role of Administrator

Task 1: Switch languages of the user interface to your desired language (English, German or Chinese)

Task 2: Building configuration

- a. Add a new floor to the building and name this new floor as "Floor 2".
- b. On Floor 2, add a bed room (name: "Bed Room"), a living room (name: "Living Room"), a kitchen (name: "Kitchen") and a toilet (name: "Toilet").

Task 3: Assign devices to the Floor 2

- a. Assign a coffee machine to the kitchen in Floor 2.
- b. Assign an air conditioner to the living room in Floor 2.
- c. Assign a light to the bed room in Floor 2.

Task 4: Add devices to the floor plan of Floor 2

- a. Upload a floor plan image (name: "floor_plan.png") for Floor 2. The image is on your desktop.

- b. Click the the coffee machine on the right panel to add it to the floor plan and drag it to the kitchen area.
- c. Click the air conditioner on the right panel to add it to the floor plan and drag it to the living room area.
- d. Click the light on the right panel to add it to the floor plan and drag it to the bed room area.
- e. Drag the circles around the device images to adjust their size.

B.1.2 Tasks for the Role of Operator

Task 1: Add a new resident to the building

The following is the personal information of this new resident. Name: test_2, Initial Password: 123456, Phone Number: 015733590751, Email: resident@gmail.com, Personal Note: created on 26.2.2018.

Now please set the following permissions for this resident.

- The allowed operations for him to use dishwasher (in kitchen) and washing machine (in kitchen) include (1) viewing these devices' general information and (2) channel information, (3) controlling these devices and (4) setting degree of freedom for these devices.
- The allowed operations for him to use freezer (in kitchen) and fridge (in kitchen) include only (1) viewing these devices' general information and (2) channel information.
- Review the above information and then submit them.

Task 2: Set optimization goals for the building

As the operator of the building, you have your own concern about energy use in your building. Therefore you want to set up some optimization goals

for your building. The energy management system of your building will balance these goals and define the best trade-off between competing goals.

Let's say, you want to minimize your energy costs, energy consumption and CO2 emissions, and you also want to consume as much electricity generated by your photovoltaic on your rooftop as possible. You have different preferences for these goals, so you set different weights to them to indicate the relative importance. Suppose

- the weight of the goal of the maximal self-consumption of photovoltaic generation is 0.2.
- the weight of the goal of the minimal costs is 0.4.
- the weight of the goal of the minimal energy consumption is 0.3.
- the weight of the goal of the minimal CO2 emissions is 0.1.

Please specify these optimization goals for your building.

Task 3: Check energy history of the building

Check energy history of the building from 09:00h, 20.12.2017 to 09:00h, 21.12.2017. Please enter the power use in this building at the time point 17:00h, 20.12.2017. If the historic data are currently not available, please enter "0".

Task 4: Building energy comparison

As an operator in the building, you want to know information about energy use in your building, and you also want to know if the energy has been used in a proper way in comparison with a community. Therefore please generate the comparison report between the average energy consumption per square meter in this building in October and the value in the community named "community_1". Check the comparison report and enter the energy consumption per square meter in this building as well as the average value in the community_1.

Task 5: Device energy comparison

As an operator in the building, you want to know whether some devices in your building are energy efficient or not when they are compared with the devices of the same type in a community. Therefore please generate a comparison report between the power use per usage for the washing machine in this building in October and the average value in the community named "community_1". Please check the comparison report and enter the power use per usage of the washing machine in this building as well as the value in the community_1.

Task 6: Check the invoice for October and enter the electricity charges you have to pay that month

B.1.3 Tasks for the Role of Resident

Task 1: Find out real-time building level energy data

- a. Find out all the devices in the building that are currently consuming electricity. Please use semicolons (;) to separate devices.
- b. Enter current voltage in the building.
- c. Check the direction of the current energy flows in the building. Is it from building to power grid or from power grid to building?

Task 2: Basic home automation

- a. Switch on the light in kitchen.
- b. Change the state of the blind in Bed Room1 to open: 100%.

Task 3: Set Degree of Freedom (DoF) for devices

Suppose you put the clothes in the washing machine at 23:00 o'clock on 24.12.2017. Since it is too late, you do not want to start the washing until tomorrow. In the morning of the next day (25.12.2017), you think

the earliest time that is allowed to start the washing procedure is at 9:00 o'clock. The washing procedure has to be finished before 14:00 o'clock. Please apply this Degree of Freedom for the washing machine.

Task 4: Check your personal energy history

As a resident living in this building, you want to know how much power you had consumed in the past. Please write down your personal energy consumption and energy cost on 20.12.2017 and compare them with those of average in the community named "community_1".

Task 5: Check energy history of a specific device

Please check the energy history of the coffee system and find out its power use at 15:00 o'clock on 20.12.2017. If the historic power use of the device are currently not available, please enter "0".

Task 6: Add a new device group and control the devices in the group by setting a state for the group

- Create a new device group and name it "All Blinds".
- Add all blinds in the building (Blind1 in Bed Room1, Blind2 in Bed Room2, Blind3 in Kitchen, Blind4 and Blind5 in Living Room) to this group.
- Set the state of the group to "closed: 25%" in order to set all blinds in the group to that state.

Task 7: Create a scene and trigger it

In the building, you can create different scenes according to your needs. Please create a scene with the following information.

- The scene's name is "evening".
- Add the lights and the blinds in Bed Room1 and Bed Room2 to the scene.

- In that scene, the target state of the lights is supposed to be on. The target state of the blinds is supposed to be closed:100%.
- After having this scene, please trigger it.

Task 8: Advanced home automation: calendar event only for devices

Add a calendar event for the building and implement the following function.

- The title of the event: blind automation.
- At 8:00 o'clock, set the state of the blind in Bed Room1 to state: open: 100%.
- At 18:00 o'clock, set the state of the blind in Bed Room1 to state: closed: 100%.
- Repeat this event every day from 24.12.2017 to 26.12.2017.

Task 9: Advanced home automation: calendar event for location and devices

If there will be some events in your building, you can mark these events on the building's calendar. Now suppose you will throw a birth party in your building. When that event starts, you hope some devices should be in some specific states. When the event is finished, you also want some devices to be in certain states. The detailed information and requirements about the event is as followed. Please add this event to the building's calendar. The energy management system in your building will take care of your settings for the event.

- The event title: birthday party.
- The start date & time of the event: 28.12.2017, 17:00 o'clock.
- Location of the event: living room.
- During the event, the temperature of the living room should be 23 °C. Disable the humidity setting. The state of the lights in the living room

should be on. The state of the blinds in the living room should be open: 100%.

- End date & time of the event: 28.12.2017, 23:00 o'clock.
- After the event is finished, disable the temperature settings in the living room. Disable the humidity settings in the living room. Set state of the lights in the living room to off. Set state of the blinds in the living room to closed:100%.

Task 10: Check the energy prediction in the building

Please enter the prediction of the basic load in the building at 19:00 o'clock this evening (22.12.2017).

Task 11: View devices' operation log

- Try to show the operation records in an ascending order by time.
- Filter the records with the keyword "light".

Task 12: Set next planned drive for the electric vehicle with the following settings.

- The departure date & time for the next drive: 9:00 o'clock on 24.12.2017.
- The distance for the next drive: 30km.
- The minimum range that has to be guaranteed for the car to drive: 20km.

B.2 Tasks for the ESHL GUI

Task 1: Find out real-time building level energy data

- a. Find out all the devices in the building that are currently consuming electricity. Please use semicolons (;) to separate devices.

- b. Enter the current net power use ("Hausanschluss") of the whole building.
- c. Check the direction of the current energy flows in the building. Is it from building to power grid or from power grid to building?

Task 2: Check energy history of the building

Please enter the power consumption ("Verbraucher") in this building at the time point 22:00h last night (20.12.2017).

Task 3: Check energy history of a specific device

Enter the power use of the washing machine at 18:00 o'clock, 20.12.2017 and enter the electricity price at that moment

Task 4: Check the energy prediction in the building

Please enter the prediction of the net power use ("Hausanschluss") in the building at 20:00 o'clock this evening (22.12.2017).

Task 5: Basic home automation

Check the state of the light in kitchen. If it is on, please switch it off. If it is off, please switch it on.

Task 6: Set Degree of Freedom (DoF) for devices

Suppose now you put the clothes in the washing machine. You want the washing procedure to be finished before 19:00 o'clock. Please specify the Degree of Freedom for the washing machine.

Task 7: Enter the remaining capacity of the battery in the building

Task 8: Enter the current voltage and frequency in the building

C Comments about the BOS UI and the ESHL GUI from the Test Users

"I find there are too many options on the left menu of the BOS UI. My feedback would be to try to regroup these options a bit."

"BOS UI might suit more people as it uses the average Google Overview as it is very intuitive."

"Maybe before using the user interfaces, there should be a tutorial first. ESHL GUI it's too complex too understand without any explanation. In BOS UI , I don't know if I already save the change I have made or not."

"BOS UI seems easy to use once you get used to it. It doesn't need much know-how to understand the interface. Whereas the ESHL GIU is not suitable for anybody like elder people who are not good with technology."

"ESHL GUI was missing the lights in kitchen."

"ESHL GUI offers too many information at once, so it is hard to get an overview of the possibilities and information."

"Music system (Multiroom) is necessary! An alarm system is optional."

"BOS UI is extremely easy to use, so that it was quite fun. ESHL GUI on the other hand is annoyingly difficult. A lot of functionalities are missing. Bad translation into German (e.g. Prädiktionen –>the word really used in German is 'Vorhersagen' or 'Prognose'), a device overview, where you also can turn them on and off, is really missing."

"BOS UI didn't work perfectly, but it looks good and tidy, intuitive to use. ESHL GUI can be a little bit confusing."

"ESHL GUI was nice animated but confusing to use. Sometimes it took me to long to get the informations. That was frustrating BOS UI: Looked not so nice but was more clear to use. Best would be a combination between these two. Nice animated and easy to use."

"ESHL GUI should be as clear and refined as BOS UI."

"Why can't I click on the time itself instead of always having to click on the little clock on BOS UI?"

"In the ESHL GUI, it's a bit harder to find and organize everything at first because there are only pictures and no labels."

"Make a submit button when the user sets things. Otherwise a lot of users will not know if it worked."

"I think BOS UI is cool. Especially the function of scene, which I also want to have in my home."

"For BOS UI, at the beginning, I am helpless. But after a few tasks, I am more and more confident to use it."

"In the overview of the ESHL GUI, I could not find the price at the 19th of December for the washing machine. I think this could be a useful add for the future."

"It is more like a nice feature, but maybe a bit over engineered. I would not set permissions for example for a party, because it is to time consuming to do that for every event."

"ESHL GUI is really unintuitive. I did not manage to find the history of yesterday, there was only the last 24 hours history. I did not like the interface. But I liked BOS UI."

"ESHL GUI is very abstract, on the other hand, BOS UI is very user friendly."

"BOS UI has a clear surface and is easy to control. It has a lot of functions. It was no problem to solve the tasks for BOS UI, but it took more time or was not possible to solve all the tasks for ESHL GUI because of the unclear user surface."

"I really like the dynamic icons in BOS UI. They give a very quick visual response of the state of a device."

"I didn't get the structure of ESHL GUI. But the problem could be that the question/tasks for BOS UI were more detailed and the description what to do was better."

"ESHL GUI is not intuitive, hard to find the needed."

"I didn't understand the ESHL GUI very well. I think, it is not very intuitive. It upset me a little bit."

"ESHL GUI didn't work properly in my opinion. BOS UI was too slow in this test."

"The ESHL GUI shows to many values. It should be kept more simple."

"I really like the intuitive BOS UI for its very accessible interface with the different folders. Idiot-proof even for beginners. It was intuitive to find everything. The floor plan was great. I like the devices were coloured orange(consuming e) or green(producing e), giving a user a quick overview of what's going on.

I had some problems with ESHL GUI, it was quite clunky to work with. The devices weren't as comfortable accessible and the charts where to find what information were kind of confusing. More or better ways to group your devices or have an overview where the most power is used at this moment, maybe even on the floor plan, that would be great."

"The ESHL GUI is not very clear. You have to make a lot more clicks to get there. Also in terms of color, the BOS UI is much better designed, which improves clarity. However, in the Energy Overview tab, for example, I find the technical parameters unnecessary. The end user is certainly not interested in what voltage is currently available."

"I find ESHL GUI very unintuitive for use. For somebody who might be already familiar with this system it might be reasonable but I had a hard time finding specific features. BOS feels much smoother, however it would be great if BOS UI can keep track of the frequency of use of my devices so that all my devices can be sorted based on the frequency of their use."

"About BOS UI:

There were several things which did not work properly:

1.) The German translation were really bad.

2.)I could not really use the Scene creator: Although there were Target States specified (sometimes even this did not work) it said that I still had to set target states.

3.)It was really unpleasant to have to click on the small arrow to set dates. It would be great if you could also click on the date itself to open the context menu.

4.)When you set things like the blinds state it would be extremely convenient to show a description of what the regulator does(e.g. on the left open, on the right closed).

5.)After using the UI for some time the computer slowed down quite heavily. I still could use everything but I don't know why a UI should generate such high load on your computer.

6.)The adding of devices was not fast enough. If this process was fastened up it would be a really great system to use.

About ESHL GUI:

1.)The GUI as such was not easy to understand. I have to admit I did not really understand what even half of the functions were displaying.

2.)All the pictures need way more descriptions!

3.)I did not really understand what the different tabs were supposed to do!?

4.)After opening a menu there should be a button to close it afterwords.

5.)The graph with the history of the electricity price was really stupid to use: After one wrong click everything displayed disappeared. I could not figure out how to go back to previous state.

6.)'Predikitionen'. Who is even using this word? Is this UI supposed to be used by customers? It did not feel this way...

–>Overall, it was extremely unpleasant to use this system because it was in no way easy to use. Nobody wants to study a UI for hours just to be able how much electricity his coffee system needs..."

"I find ESHL GUI very unintuitive for use. For somebody who might be already familiar with this system it might be reasonable but I had a hard time finding specific features. BOS UI feels much smoother, however as I've written before, it is great for checking e.g. my efficiency. For convenience there is too little of predictions or too many clicks to get to the 'sub-page' or tab to which I want to get. I have the impression that nowadays web applications have more of (what I called) 'predictions'. For example one could should a grid of all my devices sorted by frequency of usage with a field above to choose the room my device is in to filter the grid. Here I wouldn't be forced to fill out the first and then the second field to fill out the form. The same thing would happen with suggestions for the optimization, even if I don't use them they might save some time. All in all, less nested forms, more icons and grids and more user interactions & predictions."

D Task Achievement of the BOS UI and the ESHL GUI in the Evaluation Experiments

Table D.1: Task achievement of the BOS UI for the roles of administrator and operator

Task / Test User	Tasks for the BOS UI									
	Role: Administrator				Role: Operator					
	1	2	3	4	1	2	3	4	5	6
1	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
11	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
12	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
13	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
14	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
16	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
17	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
18	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
19	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
20	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Continued on next page

Table D.1 – continued from previous page

Task / Test / User	Tasks for the BOS UI									
	Role: Administrator				Role: Operator					
	1	2	3	4	1	2	3	4	5	6
21	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
22	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
23	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
24	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
25	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
26	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
27	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
28	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
29	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
30	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
31	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
32	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
33	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
34	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
35	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
36	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
37	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
38	✓	✓	✓	✓	☑	✓	✓	✓	✓	✓
39	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
40	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
41	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
42	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

P.S.: ✓: successful, ☑: partly successful, ✗: failed

Table D.2: Task achievement of the BOS UI for the role of resident

Task / Test / User	Tasks for the BOS UI											
	Role: Resident											
	1	2	3	4	5	6	7	8	9	10	11	12
1	✓	☑	✓	✓	✓	✓	✓	☑	☑	✓	✓	✓
2	☑	☑	✓	✓	✓	✓	✓	✓	☑	✗	☑	✓
3	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	✓	☑	✓	✓	✗	✓	☑	✓	☑	✗	✓	✓
7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
9	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
11	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
12	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
13	✓	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
14	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
16	✓	✓	✓	✓	✓	✓	✓	✓	☑	✓	✓	✓
17	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
18	✓	☑	✓	✓	✓	✓	✓	✓	☑	✓	✓	✓
19	☑	✓	✓	✓	✓	✓	✓	✓	☑	✓	☑	✓
20	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
21	☑	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
22	☑	☑	✓	✓	✓	✓	✓	✓	✓	✓	☑	✓
23	☑	☑	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
24	✓	☑	✓	✓	✗	✓	✓	✓	☑	✓	✓	✓
25	☑	☑	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
26	☑	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
27	✓	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
28	☑	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
29	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
30	✓	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
31	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Continued on next page

Table D.2 – continued from previous page

Task / Test / User	Tasks for the BOS UI											
	Role: Resident											
	1	2	3	4	5	6	7	8	9	10	11	12
32	✓	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
33	☑	✓	✓	✓	✗	☑	✓	✓	✓	✓	✓	✓
34	✓	☑	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
35	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓
36	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
37	✓	☑	✓	✓	✓	✓	☑	☑	✓	✓	✓	✓
38	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
39	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
40	☑	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
41	✓	☑	✓	✓	✓	☑	✓	✓	✓	✓	✓	✓
42	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

P.S.: ✓: successful, ☑: partly successful, ✗: failed

Table D.3: Task achievement of the ESHL GUI

Task / Test User	Tasks for the ESHL GUI							
	1	2	3	4	5	6	7	8
1	☑	✗	☑	✗	✗	✗	✓	✓
2	☑	✗	✗	✓	✓	✓	✓	✗
3	✓	✓	☑	✓	✓	✓	✓	✓
4	✓	✓	☑	✓	✗	✓	✓	✓
5	☑	✓	☑	✓	✓	✓	✓	✓
6	☑	✓	✗	✗	✗	✓	✗	✓
7	☑	✓	☑	✓	✓	✓	✓	✓
8	✓	✓	☑	✓	✗	✗	✓	✓
9	✓	✓	✓	✓	✓	✓	✓	✓
10	✓	✓	☑	✓	✓	✓	✓	✓
11	☑	✓	✓	✓	✗	✓	✓	✓
12	☑	✓	✓	✓	✓	✓	✓	✓
13	✓	✓	✓	✓	✓	✓	✓	✓
14	✓	✓	✓	✓	✓	✓	✓	✓
15	☑	✓	☑	✓	✓	✓	✓	✓
16	☑	✓	✓	✓	✓	✓	✓	✓
17	☑	✓	☑	✓	✓	✓	✓	✓
18	☑	✓	☑	✓	✓	✓	✗	✓
19	☑	✓	✗	✓	✗	✗	✓	✓
20	✓	✓	☑	✓	✓	✓	✓	✓
21	☑	✓	☑	✗	✓	✓	✓	✓
22	☑	✓	✗	✓	✓	✓	✗	✓
23	☑	✗	✗	✓	✓	✓	✓	✓
24	✓	✗	☑	✓	✓	✓	✓	✓
25	☑	✗	✗	✓	✓	✓	✓	✓
26	✓	✓	☑	✓	✗	✓	✓	✓
27	☑	✓	☑	✓	✓	✓	✓	✓
28	☑	✓	✗	✓	✓	✓	✓	✓
29	☑	✓	☑	✓	✓	✓	✓	✓
30	☑	✓	☑	✓	✓	✓	✓	✓
31	✓	✓	☑	✓	✓	✓	✓	✓

Continued on next page

Table D.3 – continued from previous page

Task / Test User	Tasks for the ESHL GUI							
	1	2	3	4	5	6	7	8
32	☑	✓	☑	✓	✗	✓	✗	✓
33	☑	✓	☑	✓	✗	✓	✓	✓
34	☑	✓	✗	✓	✓	✗	✓	✓
35	☑	✓	☑	✓	✗	✗	✓	✓
36	☑	✓	☑	✓	✗	✓	✗	✓
37	☑	✗	✗	✗	✗	✗	✗	✗
38	☑	✓	☑	✓	✓	✗	✗	✓
39	✓	✓	☑	✓	✓	✓	✗	✓
40	☑	✓	☑	✓	✓	✓	✓	✓
41	☑	✓	☑	✓	✗	✓	✗	✓
42	✓	✗	☑	✓	✓	✓	✓	✓

P.S.: ✓: successful, ☑: partly successful, ✗: failed

List of Figures

List of Tables

Bibliography

[1] *ACCIONA: Smart Buildings scenario definition*. http://www.fi-ppp-finseny.eu/wp-content/uploads/2012/05/D4.1_Smart-Buildings-scenario-definition_v1.1.pdf, . – [Online; Accessed: 09-01-2018]

[2] *AngularJS Data Binding*. https://www.w3schools.com/angular/angular_databinding.asp, . – [Online; Accessed: 09-05-2018]

[3] *AngularJS Material*. https://material.angularjs.org/latest/, . – [Online; Accessed: 11-05-2018]

[4] *The concept of channels in openHAB*. https://docs.openhab.org/concepts/things.htm#channels, . – [Online; Accessed: 20-03-2018]

[5] *EF-Pi*. http://flexible-energy.eu/ef-pi/, . – [Online; Accessed: 08-01-2018]

[6] *Energy consumption in households*. http://ec.europa.eu/eurostat/statistics-explained/index.php/Energy_consumption_in_households, . – [Online; Accessed: 29-03-2018]

[7] *The Energy Smart Home Lab (ESHL) at KIT*. http://www.aifb.kit.edu/web/Energy_Smart_Home_Lab, . – [Online; Accessed: 17-01-2018]

[8] *European Environment Agency (EEA) report No 17/2017*. https://www.eea.europa.eu/publications/trends-and-

projections-in-europe-2017, . – [Online; Accessed: 25-03-2018]

[9] *FHEM*. https://fhem.de/fhem.html, . – [Online; Accessed: 02-03-2018]

[10] *FHEM Tablet UI*. https://github.com/knowthelist/fhem-tablet-ui, . – [Online; Accessed: 02-03-2018]

[11] *HomeGenie*. http://www.homegenie.it/, . – [Online; Accessed: 13-06-2017]

[12] *HomeGenie website*. http://genielabs.github.io/HomeGenie, . – [Online; Accessed: 08-03-2017]

[13] *Internationalization and localization on Wikipedia*. https://en.wikipedia.org/wiki/Internationalization_and_localization, . – [Online; Accessed: 11-03-2017]

[14] *Material Design on Wikipedia*. https://en.wikipedia.org/wiki/Material_Design, . – [Online; Accessed: 25-03-2018]

[15] *Modularity on Wikipedia*. https://en.wikipedia.org/wiki/Modularity, . – [Online; Accessed: 11-03-2017]

[16] *Multilingual user interface on Wikipedia*. https://en.wikipedia.org/wiki/Multilingual_User_Interface, . – [Online; Accessed: 11-03-2017]

[17] *OGEMA*. http://www.ogema.org/, . – [Online; Accessed: 12-01-2018]

[18] *OGEMA introduction of concepts, terminology and framework services*. http://www.ogema.org//wp-content/uploads/2014/12/OGEMA_2.0_introduction_v2.0.2.pdf, . – [Online; Accessed: 19-03-2018]

[19] *OGEMA security technical notes*. https://www.ogema-source.net/wiki/display/OGEMA/Security+Technical+Notes#SecurityTechnicalNotes-WebAccessPermission, . – [Online; Accessed: 19-03-2018]

[20] *OpenHAB 1.x documentation*. https://github.com/openhab/openhab1-addons/wiki, . – [Online; Accessed: 19-03-2018]

[21] *OpenHAB introduction*. https://docs.openhab.org/introduction.html, . – [Online; Accessed: 20-03-2018]

[22] *OpenHAB Items*. https://docs.openhab.org/configuration/items.html, . – [Online; Accessed: 08-02-2018]

[23] *SAP's article about System Usability Scale*. https://experience.sap.com/skillup/quick-ux-assessment-start-with-the-system-usability-scale/, . – [Online; Accessed: 14-02-2018]

[24] *Scenes in eclipse smarthome*. https://www.eclipse.org/smarthome/documentation/features/scenes.html, . – [Online; Accessed: 13-05-2018]

[25] *The smart audio report from NPR and Edison Research*. http://nationalpublicmedia.com/wp-content/uploads/2018/01/The-Smart-Audio-Report-from-NPR-and-Edison-Research-Fall-Winter-2017.pdf, . – [Online; Accessed: 04-03-2018]

[26] *Smart home report 2017 - energy management*. https://de.statista.com/statistik/studie/id/39188/dokument/smart-home-report-energy-management/, . – [Online; Accessed: 10-03-2018]

[27] *SmartVISU*. http://www.smartvisu.de/, . – [Online; Accessed: 12-01-2018]

[28] *Technical implementation details of EF-Pi*. https://fanci.sensorlab.tno.nl/builds/fpai-documentation/development/html/, . – [Online; Accessed: 15-03-2018]

[29] *The user demo of FHEM Tablet UI*. https://github.com/ovibos/fhem-ftui-user-demos, . – [Online; Accessed: 02-03-2018]

[30] *The user demo of smartVISU*. http://demo.smartvisu.de/index.php, . – [Online; Accessed: 02-03-2018]

[31] *The user interfaces of OpenHAB*. https://docs.openhab.org/addons/uis.html, . – [Online; Accessed: 17-04-2018]

[32] *The WAMP protocol*. http://wamp-proto.org/, . – [Online; Accessed: 18-01-2018]

[33] *The WAMP protocol website*. http://wamp-proto.org, . – [Online; Accessed: 08-03-2017]

[34] *Web 2.0 on Wikipedia*. https://en.wikipedia.org/wiki/Web_2.0, . – [Online; Accessed: 24-03-2018]

[35] *Wi-SUN Alliance-interoperable communications solutions*. http://docplayer.net/55461542-Wi-sun-alliance-interoperable-communications-solutions.html, . – [Online; Accessed: 04-03-2018]

[36] ADRIAANSE, Joost ; KONSMAN, Mente J. ; COENE, G.: *High level functional specification of the flexiblePower application infrastructure*. 2013. – Technical report, TNO

[37] ALAM, Muhammad R. ; REAZ, Mamun Bin I. ; ALI, Mohd Alauddin M.: A review of smart homes - past, present, and future. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (2012), Nr. 6, S. 1190–1203

[38] ALBERT, William ; TULLIS, Thomas: *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013

[39] ALLERDING, Florian ; MAUSER, Ingo ; SCHMECK, Hartmut: Customizable energy management in smart buildings using evolutionary algorithms. In: *European Conference on the Applications of Evolutionary Computation* Springer, 2014, S. 153–164

[40] ALLERDING, Florian ; SCHMECK, Hartmut: Organic Smart Home: Architecture for Energy Management in Intelligent Buildings. In: *Proceedings of the 2011 Workshop on Organic Computing*. New York, NY, USA : ACM, 2011 (OC '11). – ISBN 978–1–4503–0736–9, 67–76

[41] BANGOR, Aaron ; KORTUM, Philip ; MILLER, James: Determining what individual SUS scores mean: Adding an adjective rating scale. In: *Journal of usability studies* 4 (2009), Nr. 3, S. 114–123

[42] BANGOR, Aaron ; KORTUM, Philip T. ; MILLER, James T.: An empirical evaluation of the system usability scale. In: *Intl. Journal of Human–Computer Interaction* 24 (2008), Nr. 6, S. 574–594

[43] BAO, Kaibin ; BRÄUCHLE, Thomas ; SCHMECK, Hartmut: Towards privacy in monitored shared environments. In: *10th Future Security Conf*, 2015, S. 469–472

[44] BAO, Kaibin ; MAUSER, Ingo ; KOCHANNECK, Sebastian ; XU, Huiwen ; SCHMECK, Hartmut: A microservice architecture for the intranet of things and energy in smart buildings. In: *Proceedings of the 1st International Workshop on Mashups of Things and APIs* ACM, 2016, S. 3

[45] BECKER, Birger: *Interaktives Gebäude-Energiemanagement*. KIT Scientific Publishing, 2014

[46] BECKER, Birger ; KELLERER, Anna ; SCHMECK, Hartmut: User interaction interface for energy management in smart homes. In: *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES* IEEE, 2012, S. 1–8

[47] BECKER, Birger ; KERN, Fabian ; LÖSCH, Manuel ; MAUSER, Ingo ; SCHMECK, Hartmut: Building energy management in the FZI house of living labs. In: *D-A-CH Conference on Energy Informatics* Springer, 2015, S. 95–112

[48] BENYON, David: Designing interactive systems: A comprehensive guide to HCI, UX and interaction design. (2014)

[49] BEVAN, Nigel: Extending quality in use to provide a framework for usability measurement. In: *International Conference on Human Centered Design* Springer, 2009, S. 13–22

[50] BHATI, Abhishek ; HANSEN, Michael ; CHAN, Ching M.: Energy conservation through smart homes in a smart city: A lesson for Singapore households. In: *Energy Policy* 104 (2017), S. 230–239

[51] BLADOW, Chad R. ; DEVINE, Carol Y. ; SCHWARZ, Edward ; SHAMASH, Arieh ; SHOULBERG, Richard W. ; WOOD, Jeffrey A.: *Graphical user interface for Web enabled applications*. September 5 2000. – US Patent 6,115,040

[52] BORODULKIN, L ; RUSER, H ; TRANKLER, H-R: 3D virtual "smart home" user interface. In: *Virtual and Intelligent Measurement Systems, 2002. VIMS'02. 2002 IEEE International Symposium on* IEEE, 2002, S. 111–115

[53] BROOKE, John: SUS: a retrospective. In: *Journal of usability studies* 8 (2013), Nr. 2, S. 29–40

[54] CLUM, Luke: A look at flat design and why it's significant. In: *Retrieved from UX Magazine: https://uxmag.com/articles/a-look-at-flat-design-and-why-its-significant* (2013)

[55] COENE, G. ; KONSMAN, Mente J. ; ADRIAANSE, Joost: *Flexible-Power application infrastructure detailed functional design*. 2013. – Technical report, TNO

[56] CONSTANTINE, Larry L. ; LOCKWOOD, Lucy A.: Structure and style in use cases for user interface design. In: *Object modeling and user interface design* (2001), S. 245–280

[57] COURCOUBETIS, Costas ; WEBER, Richard: *Pricing communication networks: economics, technology and modelling*. John Wiley & Sons, 2003

[58] DARBY, Sarah u. a.: The effectiveness of feedback on energy consumption. In: *A Review for DEFRA of the Literature on Metering, Billing and direct Displays* 486 (2006), Nr. 2006, S. 26

[59] DAVIDOFF, Scott ; LEE, Min K. ; YIU, Charles ; ZIMMERMAN, John ; DEY, Anind K.: Principles of smart home control. In: *International conference on ubiquitous computing* Springer, 2006, S. 19–34

[60] DEY, Anind K. ; ABOWD, Gregory D. ; SALBER, Daniel: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. In: *Human-computer interaction* 16 (2001), Nr. 2, S. 97–166

[61] DILLON, Andrew: The evaluation of software usability. In: *Encyclopedia of human factors and ergonomics*. London: Taylor and Francis, 2001

[62] EEKELEN, Joeri v. ; GITTE, Christian ; KAISERS, Michael ; RIGOLL, Fabian ; XU, Huiwen: *HEGRID pilot implementation*. 2015. – EIT Digital, internal, unpublished

[63] FINSTER, Soren ; BAUMGART, Ingmar: SMART-ER: Peer-based privacy for smart metering. In: *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on* IEEE, 2014, S. 652–657

[64] FRAIN, Ben: *Responsive web design with HTML5 and CSS3*. Packt Publishing Ltd, 2012

[65] GALITZ, Wilbert O.: *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007

[66] GHASEMI, Asghar ; ZAHEDIASL, Saleh: Normality tests for statistical analysis: a guide for non-statisticians. In: *International journal of endocrinology and metabolism* 10 (2012), Nr. 2, S. 486

[67] GITTE, Christian ; XU, Huiwen ; RIGOLL, Fabian ; EEKELEN, Joeri van ; KAISERS, Michael: Multi-commodity energy management applied to micro CHPs and electrical heaters in smart buildings. In: *5th D-A-CH+ Energy Informatics Conference in conjunction with 7th Symposium on Communications for Energy Systems (ComForEn)* Bd. 84, 2016

[68] GUO, Bin: *Creating personal, social, and urban awareness through pervasive computing*. IGI global, 2013

[69] HAINES, Victoria ; MAGUIRE, Martin ; COOPER, Catherine ; MITCHELL, Val ; LENTON, Fran ; KEVAL, Hina ; NICOLLE, CA: User centred design in smart homes: research to support the equipment and services aggregation trials. (2005)

[70] HARIRI, Nadjla ; NOROUZI, Yaghoub: Determining evaluation criteria for digital libraries' user interface: a review. In: *The Electronic Library* 29 (2011), Nr. 5, S. 698–722

[71] HARPER, Richard: *Inside the smart home*. Springer Science & Business Media, 2006

[72] HARTSON, Rex ; PYLA, Pardha S.: *The UX Book: Process and guidelines for ensuring a quality user experience*. Elsevier, 2012

[73] HOLZINGER, Andreas: Usability engineering methods for software developers. In: *Communications of the ACM* 48 (2005), Nr. 1, S. 71–74

[74] JANSEN, Bernard J.: The graphical user interface: an introduction. In: *SIGCHI Bulletin* 30 (1998), Nr. 2, S. 22–26

[75] KAILAS, Aravind ; CECCHI, Valentina ; MUKHERJEE, Arindam: A survey of communications and networking technologies for energy management in buildings and home automation. In: *Journal of Computer Networks and Communications* 2012 (2012)

[76] KOCHANNECK, Sebastian ; MAUSER, Ingo ; BOHNET, Bernd ; HUBSCHNEIDER, Sebastian ; SCHMECK, Hartmut ; BRAUN, Michael ; LEIBFRIED, Thomas: Establishing a hardware-in-the-loop research environment with a hybrid energy storage system. In: *Innovative Smart Grid Technologies-Asia (ISGT-Asia), 2016 IEEE* IEEE, 2016, S. 497–503

[77] LATFI, Fatiha ; LEFEBVRE, Bernard ; DESCHENEAUX, Céline: Ontology-Based Management of the Telehealth Smart Home, Dedicated to Elderly in Loss of Cognitive Autonomy. In: *OWLED* Bd. 258, 2007

[78] LOBACCARO, Gabriele ; CARLUCCI, Salvatore ; LÖFSTRÖM, Erica: A review of systems and technologies for smart homes and smart grids. In: *Energies* 9 (2016), Nr. 5, S. 348

[79] LUTOLF, R: Smart home concept and the integration of energy meters into a home based system. In: *Metering Apparatus and Tariffs for Electricity Supply, 1992., Seventh International Conference on* IET, 1992, S. 277–278

[80] MACIK, Miroslav: *Automatic user interface generation*, Faculty of Electrical Engineering Department of Computer Graphics and Inte-

raction Automatic User Interface Generation Doctoral Thesis by Miroslav Macık A thesis submitted to the Faculty of Electrical Engineering, Czech Technical University in Prague, Diss., 2016

[81] MAJUMDER, Sumit ; AGHAYI, Emad ; NOFERESTI, Moein ; MEMARZADEH-TEHRAN, Hamidreza ; MONDAL, Tapas ; PANG, Zhibo ; DEEN, M J.: Smart homes for elderly healthcare - recent advances and research challenges. In: *Sensors* 17 (2017), Nr. 11, S. 2496

[82] MANDEL, Theo: *The elements of user interface design*. Bd. 20. Wiley New York, 1997

[83] MAUSER, Ingo ; HIRSCH, Christian ; KOCHANNECK, Sebastian ; SCHMECK, Hartmut: Organic architecture for energy management and smart grids. In: *Autonomic Computing (ICAC), 2015 IEEE International Conference on* IEEE, 2015, S. 101–108

[84] MENDES, Tiago D. ; GODINA, Radu ; RODRIGUES, Eduardo M. ; MATIAS, Joao C. ; CATALAO, Joao P.: Smart home communication technologies and applications: Wireless protocol assessment for home area network resources. In: *Energies* 8 (2015), Nr. 7, S. 7279–7311

[85] MOLINA-MARKHAM, Andrés ; SHENOY, Prashant ; FU, Kevin ; CECCHET, Emmanuel ; IRWIN, David: Private memoirs of a smart meter. In: *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building* ACM, 2010, S. 61–66

[86] MÜLLER-SCHLOER, Christian ; SCHMECK, Hartmut ; UNGERER, Theo: *Organic computing-A paradigm shift for complex systems*. Springer Science & Business Media, 2011

[87] MÜLTIN, Marc ; ALLERDING, Florian ; SCHMECK, Hartmut: Integration of electric vehicles in smart homes-an ICT-based solution

for V2G scenarios. In: *Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES* IEEE, 2012, S. 1–8

[88] NIELSEN, Jakob: Do interface standards stifle design creativity. In: *Jacob Nielsen's Alertbox* (1999)

[89] NIELSEN, Jakob: How many test users in a usability study. In: *Nielsen Norman Group* 4 (2012), Nr. 06

[90] NORTHROP, Linda: The importance of software architecture. In: *Software Engineering Institute, Carnegie Mellon University. Available: http://sunset. usc. edu/gsaw/gsaw2003/s13/northrop. pdf* (2003)

[91] ORPWOOD, Roger ; GIBBS, Chris ; ADLAM, Timothy ; FAULKNER, Richard ; MEEGAHAWATTE, D: The design of smart homes for people with dementia-user-interface aspects. In: *Universal Access in the information society* 4 (2005), Nr. 2, S. 156–164

[92] PAETZ, Alexandra-Gwyn ; BECKER, Birger ; FICHTNER, Wolf ; SCHMECK, Hartmut u. a.: Shifting electricity demand with smart home technologies–an experimental study on user acceptance. In: *30th USAEE/IAEE North American conference online proceedings* Bd. 19, 2011, S. 20

[93] PASAOGLU, G ; FIORELLO, D ; MARTINO, A ; SCARCELLA, G ; ALEMANNO, A ; ZUBARYEVA, A ; THIEL, C: Driving and parking patterns of European car drivers-a mobility survey. In: *Luxembourg: European Commission Joint Research Centre* (2012)

[94] PÉREZ-LOMBARD, Luis ; ORTIZ, José ; POUT, Christine: A review on buildings energy consumption information. In: *Energy and buildings* 40 (2008), Nr. 3, S. 394–398

[95] RAYMOND, Eric S. ; LANDLEY, Rob W.: The art of unix usability. In: *Retrieved* 3 (2004), Nr. 1, S. 2012

[96] ROSCHER, Dirk ; BLUMENDORF, Marco ; ALBAYRAK, Sahin: A meta user interface to control multimodal interaction in smart environments. In: *Proceedings of the 14th international conference on Intelligent user interfaces* ACM, 2009, S. 481–482

[97] SALEN, Katie ; ZIMMERMAN, Eric: *Rules of play: Game design fundamentals*. MIT press, 2004

[98] SANDHU, Ravi S. ; COYNE, Edward J. ; FEINSTEIN, Hal L. ; YOUMAN, Charles E.: Role-based access control models. In: *Computer* 29 (1996), Nr. 2, S. 38–47

[99] SATPATHY, Lalatendu: *Smart housing: Technology to aid aging in place: New opportunities and challenges*, Mississippi State University, Diss., 2006

[100] SATZINGER, John W.: The effects of conceptual consistency on the end user's mental models of multiple applications. In: *Journal of Organizational and End User Computing (JOEUC)* 10 (1998), Nr. 3, S. 3–15

[101] SCHNEIDERMAN, Ben ; PLAISANT, Catherine: *Designing the user interface*. 1998

[102] SHEN, Bochao ; NARAYANASWAMY, Balakrishnan ; SUNDARAM, Ravi: SmartShift: expanded load shifting incentive mechanism for risk-averse consumers. In: *AAAI*, 2015, S. 716–722

[103] SMITH, Sidney L. ; MOSIER, Jane N.: Guidelines for designing user interface software / Mitre Corporation Bedford, MA. 1986. – Forschungsbericht

[104] WAAIJ, Bram van d. ; WIJBRANDI, Wilco ; KONSMAN, Mente: White paper energy flexibility platform and interface (EF-PI) / Technical report, TNO. 2015. – Forschungsbericht

[105] WILSON, Charlie ; HARGREAVES, Tom ; HAUXWELL-BALDWIN, Richard: Smart homes and their users: a systematic analysis and key challenges. In: *Personal and Ubiquitous Computing* 19 (2015), Nr. 2, S. 463–476

[106] XU, Huiwen ; KÖNIG, Lukas ; CÁLIZ, Doris ; SCHMECK, Hartmut: A generic user interface for energy management in smart homes. In: *Energy Informatics* 1 (2018), Nr. 1, S. 55

[107] XU, Huiwen ; SCHMECK, Hartmut: State-of-the-art user interfaces for building operating systems. In: *Smart Grid and Smart Cities (ICSGSC), 2017 IEEE International Conference on* IEEE, 2017, S. 283–292

[108] ZĄBKOWSKI, Tomasz ; GAJOWNICZEK, Krzysztof: Smart metering and data privacy issues. In: *Information Systems in Management* 2 (2013), Nr. 3, S. 239–249

[109] ZHOU, Bin ; LI, Wentao ; CHAN, Ka W. ; CAO, Yijia ; KUANG, Yonghong ; LIU, Xi ; WANG, Xiong: Smart home energy management systems: Concept, configurations, and scheduling strategies. In: *Renewable and Sustainable Energy Reviews* 61 (2016), S. 30–40