Xu *et al. Energy Informatics* (2018) 1:55
https://doi.org/10.1186/s42162-018-0060-0

Energy Informatics

RESEARCH Open Access

A generic user interface for energy management in smart homes

Huiwen Xu[1*], Lukas König[1], Doris Cáliz[2] and Hartmut Schmeck[1]

*Correspondence:
huiwen.xu@kit.edu
[1]Institute of Applied Informatics and Formal Description Methods (AIFB), Karlsruhe Institute of Technology (KIT), Kaiserstraße 89, 76133 Karlsruhe, Germany
Full list of author information is available at the end of the article

Abstract

Building operating systems play an important role in monitoring energy consumption of devices and improving energy efficiency in household buildings. From this arises a need for a preferably flexible and full-featured user interface to visualize the energy data in the building and allow residents to collect and realize various needs and preferences to the system. This article introduces a generic user interface for building operating systems which is presented from aspects of design, implementation and evaluation. To ensure the user interface can be flexibly adapted to various types of buildings, we design a series of generic data models which are independent of any building operating system. Besides, three roles with different permissions and a number of functional components of the user interface are also introduced in the article. Based on the design, a prototype of such a generic user interface named Building Operating System User Interface (BOS UI) has been implemented to operate the Energy Smart Home Lab (ESHL) at the Karlsruhe Institute of Technology (KIT). We evaluate the design, functionality and usability of the BOS UI qualitatively and quantitatively. The evaluation results show that the BOS UI meets a set of desired requirements (except for system configuration) for a generic user interface of building operating systems. Besides this, the evaluation experiments yielded very positive feedback in many aspects including improvement of energy efficiency and user experience. More than 90% of the test users agreed that the BOS UI provided them with enough information and functionalities that they would need in their daily lives and it can help them to save money. Furthermore, the mean score of the System Usability Scale (SUS) is 79.0, which indicates a good usability. The experiments prove that the user interface is still easy to use, despite abundant features are integrated into the system.

Keywords: Building operating system, Smart home, Generic user interface, Visualization, Energy management, Home automation

Introduction

Nowadays improving energy efficiency has become a global concern in terms of saving natural resources and cutting down on pollution. During the last two decades (1984-2004) primary energy consumption has grown by 49% and CO_2 emissions by 43%, with an average annual increase of 2 and 1.8% respectively (Pérez-Lombard et al. 2008). Energy use in buildings represents about 40% of the EU's total final energy consumption and CO_2 emissions (Balaras et al. 2007), therefore improving energy efficiency and making best use of renewable energies in buildings will play a significant role in mitigating the global energy

and climate crisis. To achieve this, a lot of research focuses on design and implementation of building operating systems or energy management systems. For instance, HomeOS (Dixon et al. 2012) provides a PC-like abstraction for home technology. The Organic Smart Home (OSH) (Allerding and Schmeck 2011) helps to optimizes the schedule of appliances based on evolutionary algorithms in order to help residents to save money. In Mauser et al. (2016), the OSH has been extended to support multi-commodity energy management in smart homes.

A user interface is considered an essential part of a building operating system in a smart home since it is supposed to help keeping residents in the loop by providing them with useful information and interactive options. In Orpwood et al. (2005), design methodologies and evaluation approaches of user interfaces for people with dementia in smart home environment are elaborated. There is, however, no user interface prototype that is implemented, based on the methods mentioned in this article. In Borodulkin et al. (2002), a working prototype of 3D virtual graphical user interface for a smart home is described. In the prototype, a sophisticated 3D virtual smart home environment has been designed by using a Virtual Reality (VR) approach. The virtual reality user interface enables residents to view consumption values and manipulate appliances. Nevertheless the evaluation of the prototype is missing in the paper. In Becker et al. (2012), a user interface for providing the connection between the resident and the energy management named Energy Management Panel (EMP) is presented. The EMP is designed specially for the Energy Smart Home Lab (ESHL) (The Energy Smart Home Lab at KIT). It is evaluated in the scope of test residents by students at the KIT. Macık (2016) proposes a method for context-sensitive automatic user interface generation which has been showed the possibility of supporting solutions for the domain of the Internet of Things. However when it comes to put the theories into practice, namely generating a user interface for a specific building operating system, the author claims that there are still many challenges which need to be further addressed, therefore there is no available user interface instance for building operating systems in the article.

Most of the research about user interfaces for building operating systems in smart homes done in the past years (e.g. Orpwood et al. (2005), Borodulkin et al. (2002), Macık (2016), Latfi et al. (2007) and Roscher et al. (2009)) have dealt primarily with method description, which is more theoretical in nature. So far, we are not aware of many publications in the area of practical implementations of such user interfaces. In this article, a generic and flexible user interface for building operating systems is designed, implemented and evaluated, focusing mainly on the aspects of energy management and comfort. The user interface, on the one hand, is able to communicate with building operating systems in various smart homes, on the other hand, it can provide residents with holistic and transparent information on energy consumption and generation in their building as well as control over appliances. In Xu and Schmeck (2017), a number of popularly open-sourced user interfaces for building operating systems were evaluated from the aspects of smart home related use cases and some technical characteristics. The results showed that all the user interfaces that were evaluated have space to be enhanced in varying degrees from different aspects. They either tightly couple with a specific building operating system, or only cover limited use cases about smart homes.

The remainder of this article is structured as follows. "Design" section clarifies some concepts that will be later used in this article and then proposes an architecture for a

generic user interface for building operating systems. In order to achieve the goal of making the user interface generic, we introduce different roles with different permissions, design some properly abstracted data models and come up with a number of functional components to support a wide range of use cases relating to smart homes. Based on the design in "Design" section, a prototype of the generic user interface has been implemented as described in "Implementation" section . The prototype is evaluated in terms of design and usability. "Evaluation" section presents the according results and discusses the relevant implications. In "Discussion" section, we summarize key features and novelty of the current research, and point out challenges of implementing our concept in an actual household building. Finally, we conclude the article and provide an outlook on future work in "Conclusion and outlook" section.

Design

With a rising number of competing building operating systems in the future market, having a holistic and comprehensive overview of the energy flows and devices will become more and more difficult. In a fragmented market, the customers might lose control over their individual targets concerning their building. This section presents the design of a generic user interface for building operating systems, which can be used for future user interfaces to deal with heterogeneity of different building operating systems.

Definitions

During the 1990s the concept of the smart home entered popular culture for the first time, which aroused great public awareness (Harper 2006). Henceforward this concept has gradually become a reality with the rapid growth of the internet of things (IoT). Since entering the 21st century, smart home or home automation technologies have been increasing in popularity and began changing the way people live. At the same time a variety of smart home related terminologies arose which are widely used but not uniformly defined. For the sake of clarity, terms and expressions used in this article are defined in the following:

A **Household Building** refers to either one of the apartments in a residential building or an independently single or multi-storied house which is owned by one person or family. An important feature for the "building" in this article is that there is a person who is responsible for paying all the energy costs. Residential buildings that are shared by a number of different home owners, office buildings and other public buildings are not in the scope of this article, but but the concepts presented in this article can be extended to these scenarios.

A **Building Operating System** is an Information and Communication Technology (ICT) system that is deployed in a household building. Similar to computer operating systems, such as Windows or Linux, which coordinate hardware components and provide services for applications software, a building operating system manages heterogeneous household devices, sensors, storages and other equipment by means of IoT technologies, and on the other hand, it optimizes energy use in the building and provides various services for residents.

A **Generic User Interface for Building Operating Systems** (which is the work basic that has been done in this article), as a general term, has different commonly accepted definitions. It can also apply to a building operating system, in which case, its concept is still

self-evident. However, when it comes to a **generic** user interface for building operating systems, so far there is no available definition for this concept in the existing publications. In this article, 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 specific hardware or operating system (Bladow et al. 2000). 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. A web-based user interface, on the other hand, is flexible to be accessed from any computer connected to the Internet using a standard browser, that is why it is widely getting popular.
- **Responsiveness**. A generic user interface should be responsive. According to the survey conducted by the Pew Research Center in 2015 (U.S. Smartphone Use in 2015), 68% of Americans are smart phone owners, up from 35% in 2011, and tablet ownership has edged up to 45% among adults. This growth continued through these years. With the increasing number of mobile devices with different screen sizes, it is important to create a responsive user interface by using the Responsive Web Design (Frain 2012) which can assure 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 (Galitz 2007).
- **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 (Constantine and Lockwood 2001). When it comes to the field of smart buildings or smart homes, although there are usually not so many residents living 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 recognise that users value time, roles and relationships in their domestic lives, there are growing concerns that smart home technology is coming to dominate people, rather than the other way around (Wilson et al. 2015; Davidoff et al. 2006). A role-based user interface is able to assign users to different roles which have different permissions to access the system, leading to a higher protection of privacy and a better acceptance of users. The restricted access to data or certain functionalities ensures confidentiality, integrity, and accountability (Guo 2013).
- **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 applicable scope of a user interface, which will affect the applicability and generality of the user interface to a great extent. Xu and Schmeck (2017) reviewed the state-of-the-art user interfaces for building operating systems and then came up with a set of common smart home related use cases, which can be used as a reference for the functional design of the generic user interface for building operating systems. We refer to this set of use cases when discussing generality in the remainder of this article.

Apart from eliminating ambiguity, the requirements listed above, on the one hand, identify guidelines for the design of such a generic user interface in this article, and, on the other hand, provide criteria to evaluate the user interface prototype implemented on the basis of the design (cf. "Evaluation" section).

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 (EMP) (Becker et al. 2012) is the user interface of the OSH (Allerding and Schmeck 2011). EF-Pi (EF-Pi) provides some empty widgets, that need to be developed by application designers, as its user interface. OpenHab2 (OpenHAB) uses the Paper UI, the Basic UI and the Classic UI (The User Interfaces of OpenHAB 2) as its user interfaces to realize different home automation goals. Since the smart home market, at this stage, is still highly fragmented, the existing building operating systems usually use different standards and provide different functionalities, with the result that user interfaces are heavily coupled with their corresponding building operating systems, and these user interfaces either support only a limited number of use cases relating to smart homes or are not flexible enough 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 1 shows the architecture of such a generic user interface. The key part that makes the user interface generic and extendible is a number of generic data models, which are appropriate abstractions for objects that are needed by the generic user interface. Part of the data models will be explained in "Data models" section. The existing building operating systems (e.g. OSH, EF-Pi, or another random building operating system named BOS A, which are represented by the first three dotted boxes in Fig. 1) 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. 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

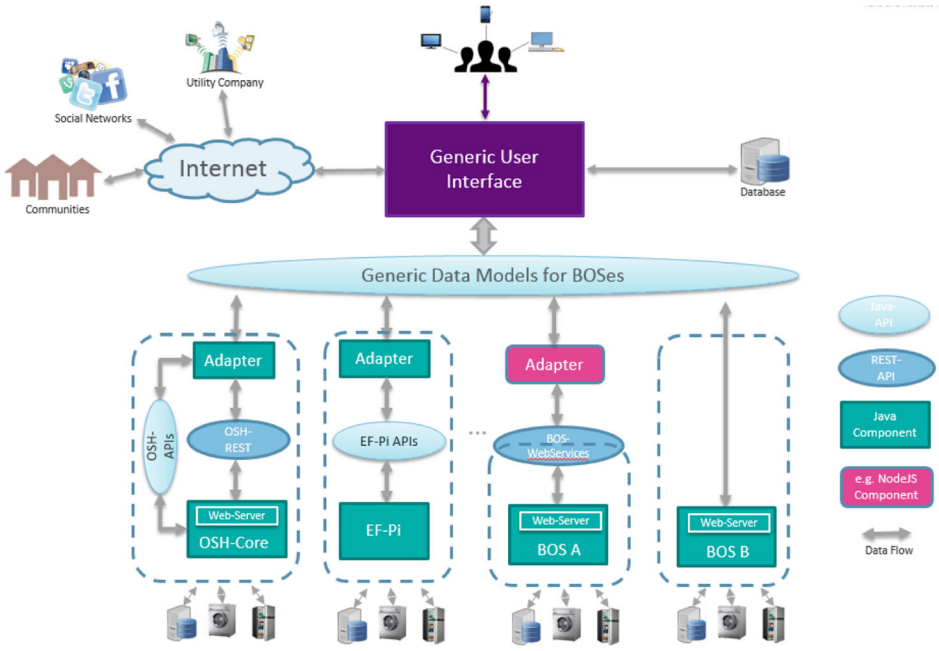


Fig. 1 Architecture of the generic user interface for building operating systems in smart homes

system, which is supposed to be neither logical nor realistic. The adapter can be implemented either inside 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 article 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.

Furthermore, building operating systems in 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 clear that the generic user interface for building operating systems is able to connect to a utility company to not only receive regular energy bills but also access customized services made by residents. If the energy information from utility companies are available as structured data instead of the usual files (e.g. PDF files), in other words, if the energy data are organized with standard data models, there will 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 young generation, desire to share their lives on social media, e.g. Facebook and Twitter. For this reason, it would be beneficial for residents to share their energy achievements in their building with their friends on popular social networks via the generic user interface. This would also be a disguised incentive for users to improve their awareness for energy conservation.

Finally, with the rising number of smart meters being rolled out in all kinds of buildings, 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. 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. Xu and Schmeck (2017) and ACCIONA: Smart Buildings Scenario Definition). 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 (Molina-Markham et al. 2010). 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 (Zabkowski and Gajowniczek 2013). In addition, some privacy-preserving smart metering protocols (e.g. the SMART-ER Protocol (Finster and Baumgart 2014)) could also be used to prevent from privacy violation of residents.

Roles

In one of the deliverables, Smart Buildings "scenario" definition (ACCIONA: Smart Buildings Scenario Definition), from the FINSENY project (FINSENY), the home dweller which represents all categories of persons who live in a home permanently is thought of the only home domain actor. Even the authors point out, however, 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: The homeowner is not comfortable if visitors can see all the energy data in his building, so he wants to restrict the amount of information displayed on the user interface.

Scenario 3: The homeowner owns the appliances in a building but he is not living in the building. Instead, he rents the rooms out to different tenants who do not want the homeowner to track their energy use, in consideration of privacy protection.

Scenario 4: The homeowner owns the appliances in a building and he is living in the building, but he rents the other rooms, which he does not need, to other tenants. So he can monitor his own energy use in the building through the user interface which is the same as the other tenants. But he is also able to view more global energy use in 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 brought by various scenarios, in this article, the users of the generic user interface are classified into three roles: the administrator, the operator and the resident. Figure 2 shows the relationship between these three roles.

The administrator is in charge of the configuration of a building, e.g. assign rooms to the virtual building within the user interface and devices to corresponding rooms. The resident is the role ascribed to users who are living in the building and use the appliances on a daily basis. The operator is the role held by the one who needs to pay the energy bills for the building. Besides this, he is also responsible for managing 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 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 article 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.

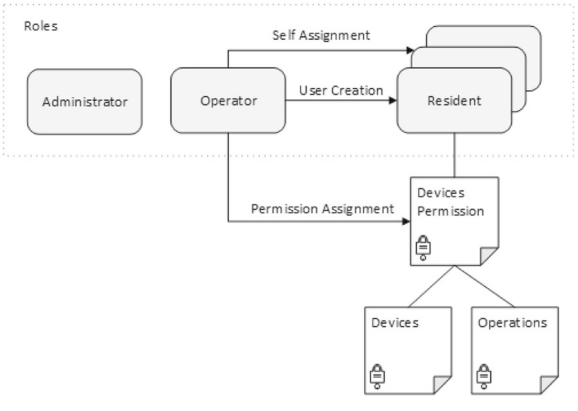


Fig. 2 The three user roles of the generic user interface for building operating systems in smart homes

- **Set Degree of Freedom**. The degree of freedom of household devices refers to the time interval within which the devices can be re-scheduled (Paetz et al. 2011). The working schedules of some white goods, e.g. washing machine or dishwasher, 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 for the aforementioned operations is progressively increased (cf. Fig. 3). If one can view device channel information, it implies he can also view device general information. If one has the permission to control the device then he can view both general information and channel information of the device. Similarly, the permission of Set Degree of Freedom covers the rest of operation permissions.

Under the condition that the operator is also living in the building as one of the residents, 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 ideas of the roles in this article were inspired by Role-Based Access Control Models (Sandhu et al. 1996), where the author defined a family of four conceptual models for various dimensions of Role-Based Access Control (RBAC). The models in Sandhu et al. (1996) have a common assumption that there is a single security officer who is the only one authorized to manage RBAC. However, the context of household buildings should be different. It is not reasonable to have such a chief security officer for the sake of personal privacy protection. 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 costs in the building, is denied access of 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 also possible for one person to have the two roles at the same time. In this case, it means there is

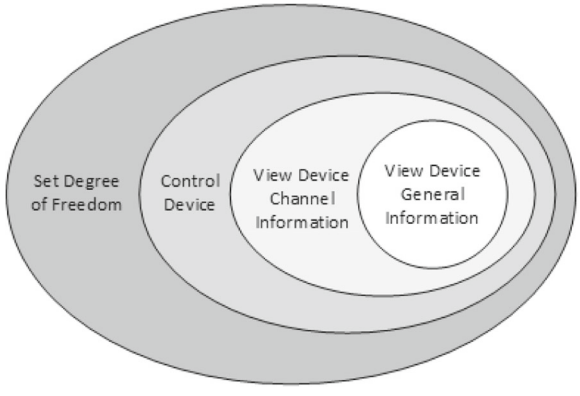


Fig. 3 The permission hierarchy of the four operations for devices

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.

Data models

As the central part of the architecture of the generic user interface for building operating systems (cf. Fig. 1), 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, which is the case in Becker et al. (2012). The problem of these data models is that they are not flexible to extend to other buildings. EF-Pi (EF-Pi) and OGEMA (OGEMA) do not even provide data models to their user interface. They leave the tasks to their application designers. In openHAB, the functionalities that are used by its user interfaces and automation logic are abstracted into so-called Items (OpenHAB 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 washing machines, 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 for openHAB is that the designer of its user interfaces needs to know how the Items are defined in the openHAB instance, which means, the development of the user interface is dependent on the definition of the system instance. Similarly, with the aim to visualize KNX home automation systems, the data models provided by smartVISU (SmartVISU b) only cover five sorts of devices.

In this article, a generic data model for devices in household buildings is designed in the scope of the generic user interface. Its UML class diagram is shown in Fig. 4. The class HouseholdDevice is derived from the Device class, which consists of basic information about a device, e.g. a device identifier, a name and a list of device factory information. In addition to the attributes inherited from its base 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 article, refers to a functional component of a device. Most household devices, e.g. lights and televisions, provide a single function, so they have only one channel, which can display devices' working states, power use, or some other information on the generic user interface. Some devices are more complex so that they need more than one channel to display their full information. For instance, the 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 the boiler and the power engine are abstracted to two channels for the μCHP. Sometimes the μCHP can 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 about

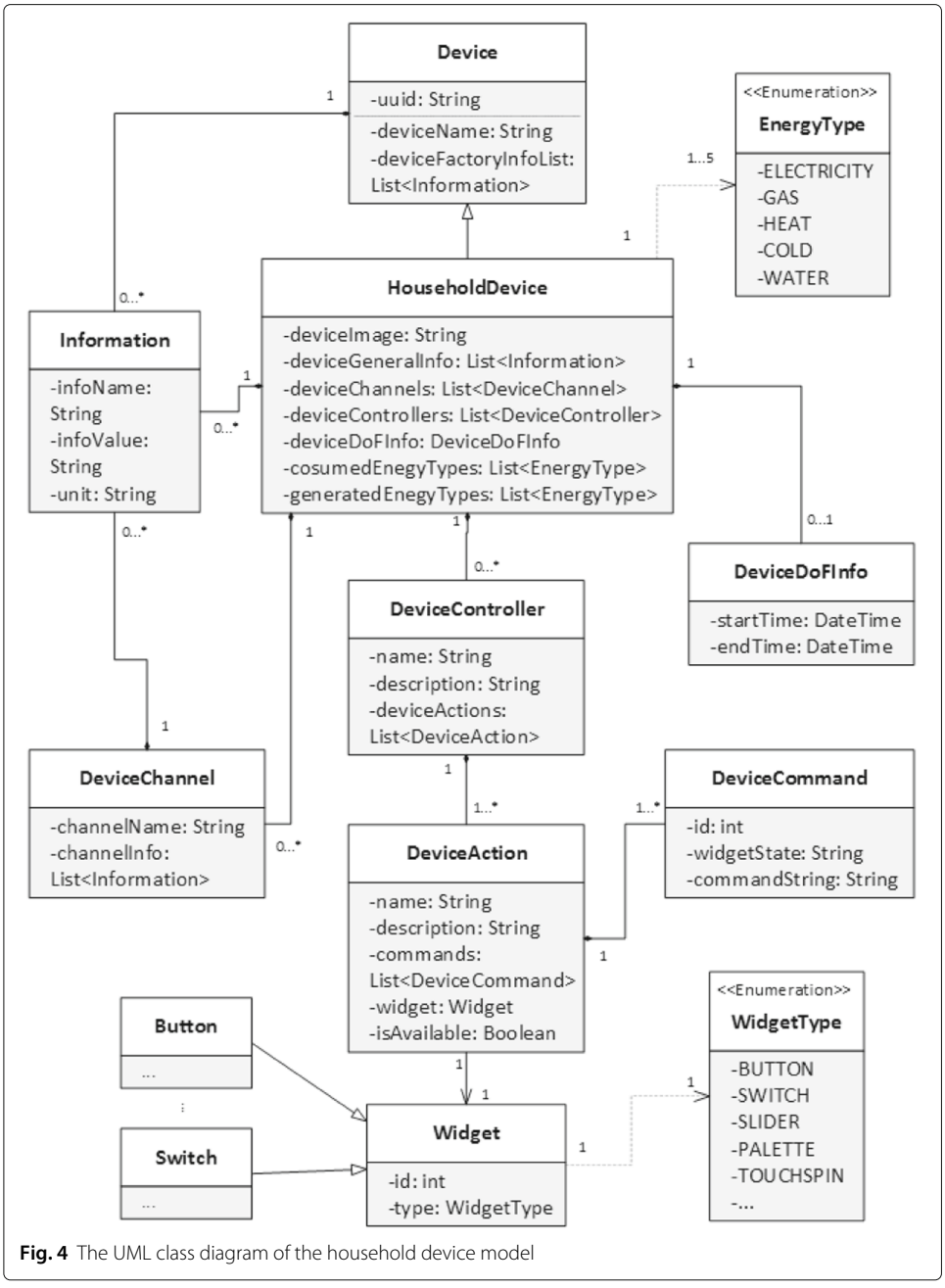


Fig. 4 The UML class diagram of the household device model

the device. The information about a channel (channelInfo) are put into a list of Information objects. Each object consists of an information name, an information value and a unit about the information. For instance, one information about the boiler channel of a μCHP could be described as "infoName: top temperature, infoValue: 69.5, unit: °C". The information are provided by the underlying building operating systems and further will be displayed on the generic user interface.

Besides information about the device channels, there is some globally general information about the device which is not specific to a single channel, e.g. the location of the device and the list of residents who are allowed to access the device, etc. This general

information about the device is covered by the field of "deviceGeneralInfo", which also consists of a list of Information objects, the same as channelInfo.

In order to facilitate the classification of devices on the generic user interface, the consumed and generated energy types (electricity, gas, heat and cold) are covered by the fields of "consumedEnergyTypes" and "generatedEnergyTypes", respectively.

In addition to displaying information about the device on the generic user interface, the HouseholdDevice data model provides means to interact with devices. For devices having a degree of freedom, their allowed starting time and required end time, which is specified by residents on the user interface, are covered by the field of "deviceDoFInfo". Except for sensors, most of the household devices provide one or more functions that can be controlled by residents. Every function unit is abstracted into a so-called DeviceController, therefore the HouseholdDevice model contains a list of device controllers. For example, some multifunctional air conditioners can regulate and control the temperature, humidity and cleanliness of the air, therefore, in this case, each of the functions provided by the air conditioners corresponds to a DeviceController that contains a list of DeviceActions. Each DeviceAction has a Widget (e.g. button, switch, slider, etc.) and a list of DeviceCommand relating to the Widget. The DeviceCommand records what a command (commandString) will be sent to the building operating system in what state of the Widget (widgetState). When the state of the Widget is changed by residents, the generic user interface will find the corresponding command string according to the widget state and send it to the building operating system so as to realize the control of the device.

A household building in this article 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 5 shows the UML class diagram of the floor model. The floor model represents the global configuration of a building, therefore it will be managed by the administrator via the generic user interface.

A Floor is made up of a list of Locations (e.g. rooms or hallway). Each Location has a name and a list of household devices, whose data model is shown in Fig. 4. Besides this, 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 article is to display the users' household devices on the floor plans of their building. A FloorPlan class (Fig. 5) 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. 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. 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.

As one of the basic use cases in the context of smart homes, simple home automation, such as changing the state of a specific device (e.g. switching a light on or off), can be implemented based on the HouseholdDevice model (cf. Fig. 4). In a smart home, the generic user interface is supposed to enable residents to realize advanced home

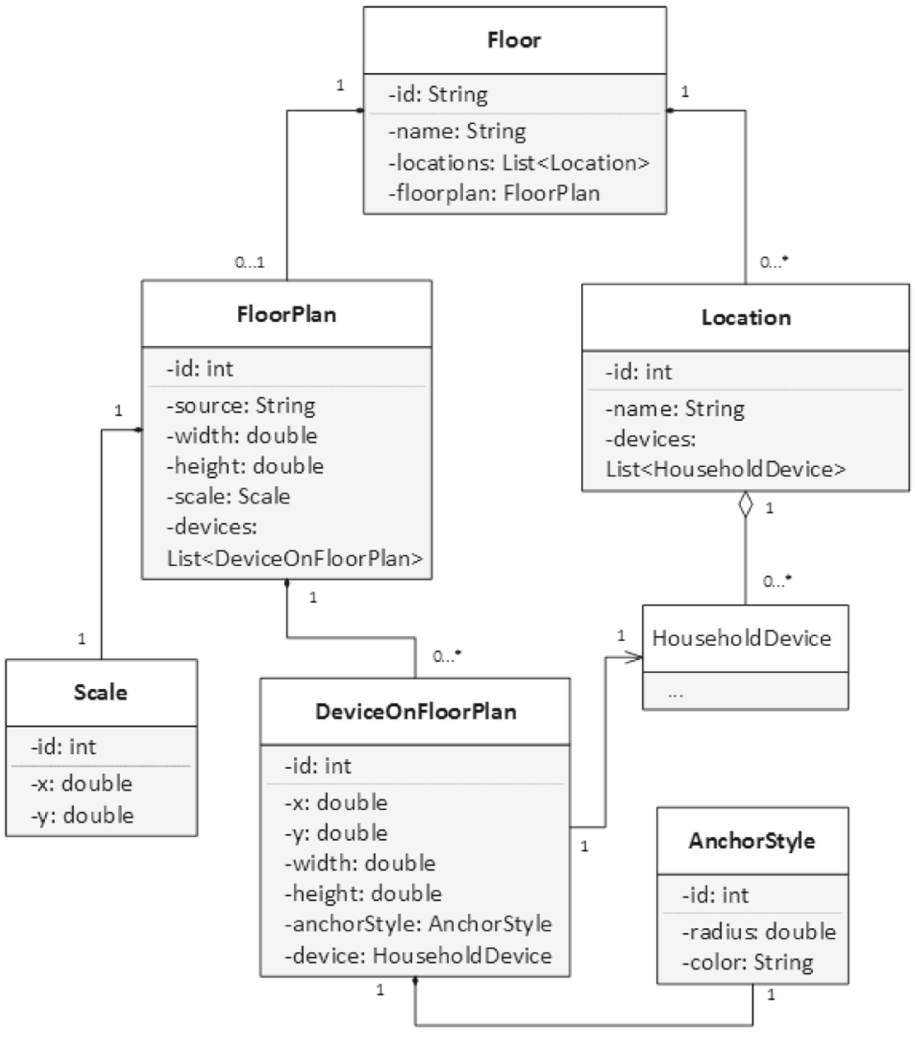


Fig. 5 The UML class diagram of the floor model

automation. For instance, the generic user interface enables residents to create various scenes in their building according to their needs. The class diagram of the scenes is shown in Fig. 6.

A scene is owned by a resident who created it on the generic user interface. The resident can add devices, which they have permissions to control, to the scene. A device that has been added to the scene is presented by the class DeviceInScene which is associated with a household device. If a household device is controllable, its data model HouseholdDevice will contain one or more DeviceController members. A DeviceController further has one or more DeviceAction members. Each of them contains a widget and a list of DeviceCommand instances which consist of different widget states and their corresponding command strings that need to be sent to the building operating system in order to control devices to reach the states in reality.

On the generic user interface, the resident can specify the target states for the devices in the scene, which together with their command strings will be saved by a list of TargetStateAndCmd instances (i.e., targetStatesAndCmds). The TargetStateAndCmd class

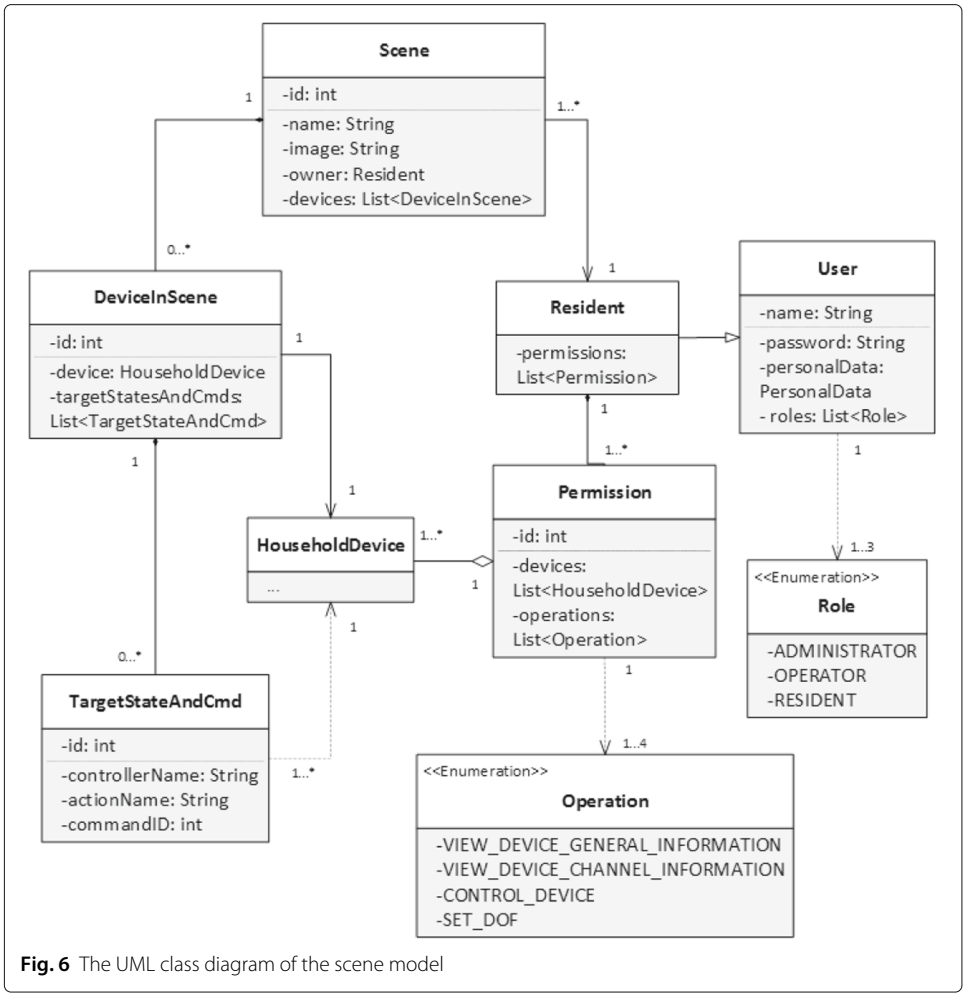


Fig. 6 The UML class diagram of the scene model

contains the identifier of the DeviceController instance (i.e., controllerName) of the household device, the identifier of the DeviceAction instance in that DeviceController (i.e., actionName) and the identifier of the DeviceCommand instance in the DeviceAction. Through these three identifiers, the generic user interface can easily find the device's target state, that has been set by the resident, and the corresponding command string, that 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 target state. The detailed process of adding and triggering a scene can be found in Fig. 7.

One of the most important features that the generic user interface is supposed 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 organise the household devices on the basis of their locations in the building. There is no doubt that

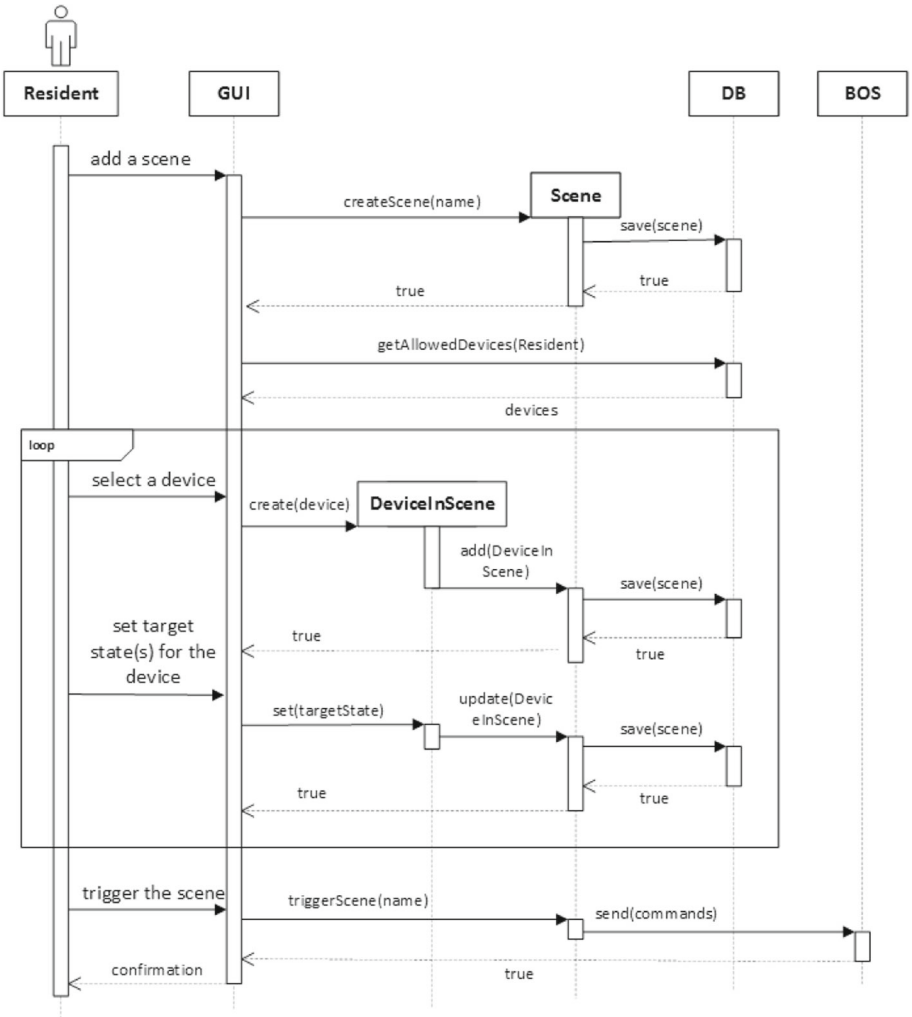


Fig. 7 The sequence diagram of adding and triggering a scene

organising devices in this way is logical and intuitive for residents to find devices. However, it is not an intuitive way for residents to get a whole picture of the energy use in their building since the different kinds of devices that are running are scattered in many places. Thanks to the household device model (cf. Fig. 4), which contains consumed energy types and generated energy types as its attributes, the generic user interface is able to provide a different perspective (cf. Fig. 8) to organize devices in order to help residents to know what is going on in their building.

The in-house energy overview provided by the generic user interface is classified in multiple levels which are organized into a tree structure. The first level is based on the energy types, which include electricity, heat, cold, gas and water. Under each energy type is the second level which consists of various running modes of the different devices for this energy type. For instance, under the energy type of electricity, the devices are classified into three types: power consumption (e.g. lights, TVs, etc.), power generation (e.g. photovoltaic panels) and storage (e.g. batteries). The third level refers to either devices or

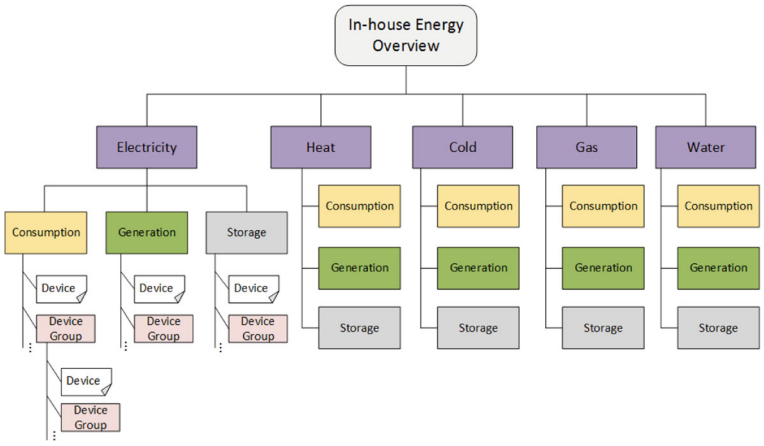


Fig. 8 The data structure of energy overview provided by the generic user interface

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 can be defined to contain all of the lights in the building. Within the light group, a nested group named "living room" can be further defined to contain all the lights in the living room.

By fragmenting the whole "level" of the energy use 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 generic 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.

Functional components

Based on the requirements of the generic user interface for building operating systems, this section introduces the detailed functional components (cf. Fig. 9) that are supported by the generic user interface.

As described in the previous section, three roles are created for various responsibilities in the household building, namely, the administrator, the operator and the 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. After 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 for different residents, which means that the residents can be restricted to use certain devices with only authorized operations. Since the operator needs to pay the energy bills, besides checking the invoices sent from the utility company on the generic user interface, he can 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 reasonable for the operator to set multiple (even conflicting) optimization goals by having him specify weights for different goals. The building operating system is supposed to balance these goals and define

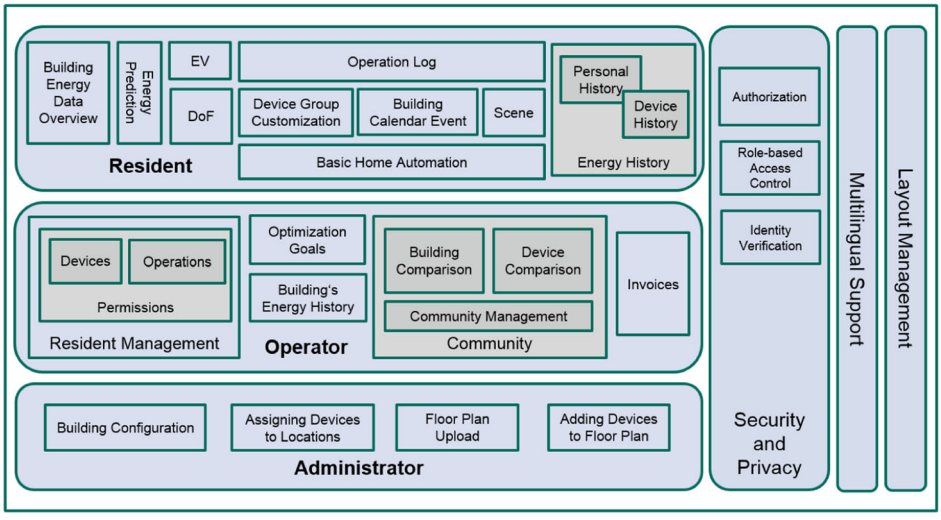


Fig. 9 The functional components of the generic user interface for building operating systems

a fairly well trade-off between competing goals. In order to get a better view of the energy use in the building, the operator can check the building's energy consumption and generation history by taking electricity prices and load limits as reference. What is more, the operator can choose to join some "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 in the building. As for the historical energy information, residents can view both their personal historical energy use and historical data for a specific device. Besides, they can configure the next drive for the electric vehicle and set degrees of freedom for devices. In addition to basic home automation, the generic user interface supports residents to execute advanced home automation which makes possible to execute multiple actions at specific time, location, or event to manage the building in a smart way.

Firstly, it allows residents to customize their own device groups. For instance, residents can group all the lights in a building to one light group, so that they can control all the lights by sending commands to the group. Additionally, a resident can create different scenes according to his needs. He can add devices to a scene and specify target states for the devices via the generic user interface. After completing the configuration, the scene can be triggered any time by the resident. Furthermore, the generic user interface provides

a calendar service which plays an important role in energy optimization in the building. The residents' schedule information, marked on the calendar, can be used as auxiliary information for the building operation system to do optimization. From the residents' perspective, the calendar service enables them to realize advanced home automation by adding calendar events. For example, a resident can add an event on the calendar to roll up the blinds in the building in the morning and roll down them in the evening for a specific day or repeating it every day. Another example is having a party which is a more complicated event. To this end, on the calendar, the resident, on the one hand, can set parameters (e.g. temperature or humidity) for the location where the event will take place. On the other hand, he can select the devices that are needed for the event and set target states for the 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.

In terms of the system as a whole, the generic user interface provides authorization, role-based access control and identity verification to ensure security and privacy. Moreover, it provides multilingual support and layout management to ensure a good user experience.

Implementation

Based on the design in the previous section, a prototype of the generic user interface has been implemented and applied to a building operating system, the Organic Smart Home (OSH) (Allerding and Schmeck 2011), which is based on the Observer/Controller architecture in Organic Computing (Schmeck 2005). With the aid of evolutionary algorithms, the OSH optimizes the schedule of appliances so as to minimize energy costs for residents. Since 2009, the OSH has been deployed at the Energy Smart Home Lab (ESHL) (The Energy Smart Home Lab at KIT) of the Karlsruhe Institute of Technology (KIT).

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 prototype of the user interface that has been implemented is named BOS UI, which is short for Building Operating System User Interface. The communication among the BOS UI, the OSH, and the devices in the ESHL is achieved via a WAMP router. The relationship and the communication between these components is shown in Fig. 10.

The dotted box in Fig. 10 illustrates the communication mechanism between the OSH and the devices in the ESHL. It is implemented by means of the Web Application Messaging Protocol (WAMP) (The WAMP Protocol), which is an open standard WebSocket subprotocol that provides two application messaging patterns in one unified protocol: Remote Procedure Calls (RPC) and Publish & Subscribe (PubSub). The OSH is using a WAMP router as a massage bus to communicate with the devices in the ESHL. Within

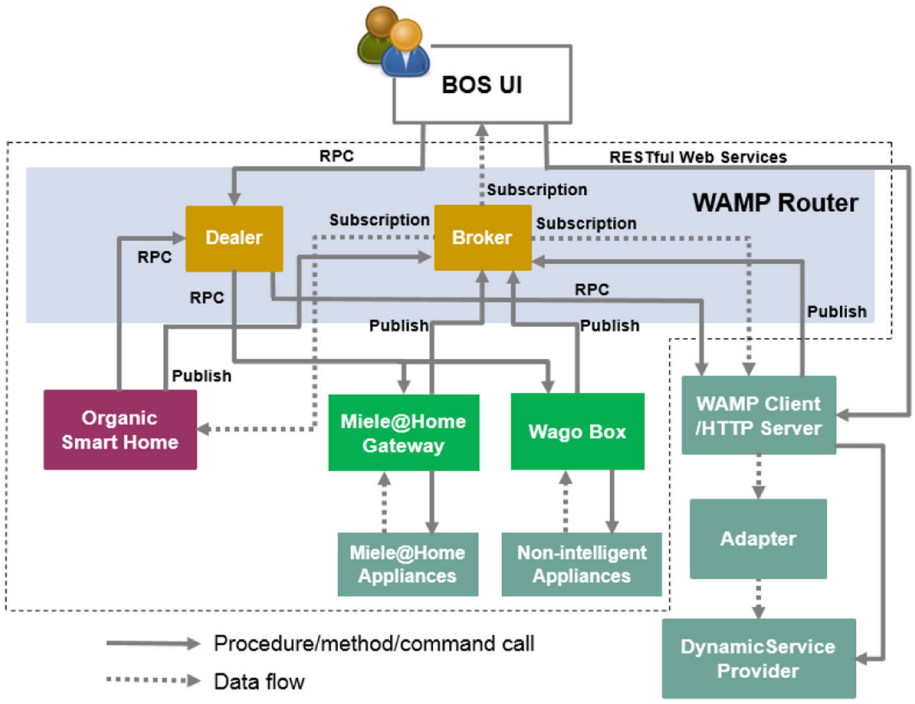


Fig. 10 The communication between the BOS UI, the OSH, the devices in the ESHL via a WAMP router

the WAMP router, two components, a Dealer and a Broker, are used as intermediaries to route PubSub events and RPC calls between the OSH and the devices in the ESHL. The Broker keeps a book of subscriptions so that it will forward the updated data of different topics to the OSH. Similarly to the Broker's role to PubSub, the Dealer keeps track of the procedures that have been registered in the WAMP Router, so the OSH can call the registered remote procedures from Dealer which will help to invoke the procedures. 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. The data provided by these topics can be shared with the OSH and the BOS UI by subscribing to corresponding topics of interest.

In order to apply the BOS UI to the ESHL, three more components have been added to the current system in the ESHL.

- **The WAMP Client/HTTP Server** is a component that includes two functions. On the one hand, it acts as a WAMP client which communicates directly with the WAMP Router to access real-time changing data (e.g. power values of devices) in the ESHL by making subscriptions, publishing new topics and registering remote procedures to the WAMP Router. The BOS UI will subscribe the topics and call the procedures to access real-time data in the ESHL from the WAMP router. On the other hand, it acts as an HTTP Server for the RESTful Web Services, which are called by the BOS UI in order to retrieve the information in the ESHL that are not updated frequently (e.g. building configurations).

- **The Adapter** receives data from the WAMP client as the input and converts them into the standard data models (cf. "Design" section) consumed by the BOS UI as the output.
- **The DynamicServiceProvider** provides all kinds of integrated services 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.

With the aid of the aforementioned components, the BOS UI is able to communicate with the OSH and the devices in the ESHL. As for dynamically updated data (e.g. device states) or various historical energy data of the ESHL, the BOS UI can retrieve them 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. For the data which are not updated dynamically or frequently in the ESHL, they can be accessed and updated by the BOS UI by means of a number of RESTful Web Services.

The BOS UI is developed on the basis of Fuse (Fuse), which is an AngularJS template that uses Angular Material library. AngularJS Material is both a UI Component framework and a reference implementation of Google's Material Design Specification (AngularJS Material). Benefiting from these technologies, on the one hand, the BOS UI is equipped with a responsive layout that can adapt to screens with different sizes. On the other hand, it is capable of producing a rich and compelling visual experience for users.

In view of the fact that the users of BOS UI are classified into three roles, namely, the administrator, the operator and the resident, the BOS UI also takes on three different views for these three roles.

The major responsibility of the administrator is to configure the building. The user interface for the administrator (cf. Fig. 11) consists of three menu items. The first item is "Manage Locations", where the Administrator can add floors to the building and define locations for the floors. The second menu item is "Assign Device to Floor", where the administrator can assign the devices in the building to proper locations that have been defined in the first menu item. The last menu item is "Add Device to Floor Plan"

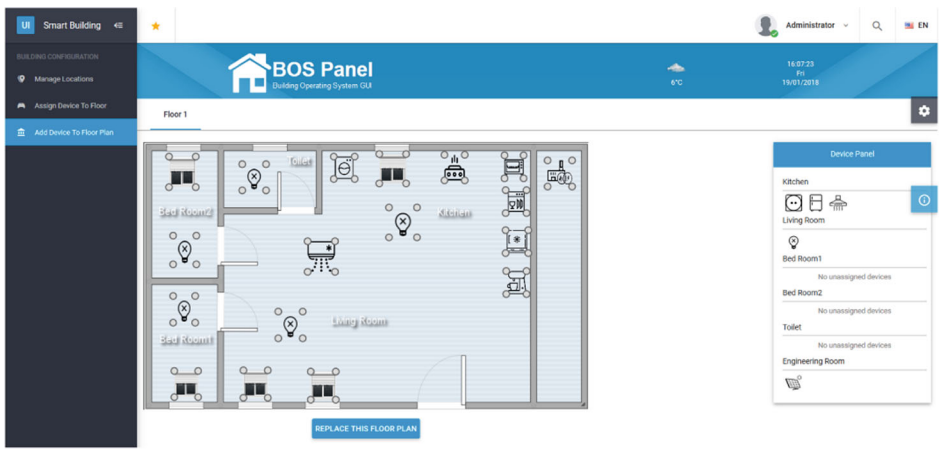


Fig. 11 BOS UI for the role of Administrator

(cf. Fig. 11). Under this menu item, the administrator can upload floor plan images for the floors that have been added in the first menu item. The device panel on the right shows all of the devices that have not been 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.

For the operator, the BOS UI provides options for him to manage residents in a building. To add a resident to a building, he needs three steps to complete the operation. Firstly, he needs to specify the resident's personal information including username, initial password, phone number, email, etc. The second step is to assign permissions to the resident. Every permission consists of a list of devices that are allowed to be used by the resident and permissible operations of these devices. The available operations include "View Device General Information", "View Device Channel Information", "Control Device", and "Set Degree of Freedom". The last step is to review all the information that have been set in the last two steps and submit the information to the server in order to create an account for the resident.

Besides this, since the operator is responsible for paying energy bills in a building, he can set optimization goals for the building. In the BOS UI, four optimization goals are currently provided. They are Minimal Costs, Minimal Energy Consumption, Minimal CO_2 Emissions and Maximal Self-consumption of Renewable Generation. Each of the goals is presented with a slider, with which the operator can set weight (within the range of 0 to 1) for the goal to indicate its relative importance. The competing goals set by the operator is supposed to be balanced according to their weights by using multi-objective optimization methods in the building operating system. It is noteworthy that the optimization goals formulated in this article 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.

In order to get a clear view about the building's energy use in the past, the BOS UI enables the operator to check the energy history of the building (cf. Fig. 12). On the BOS UI, the operator can specify a starting time and an end time for the history. The historical data, that can be viewed, include Building Power Consumption, Building Power Generation, Building Net Power Use, Electricity Price, Load limit, PV Feed-in Price and μCHP Feed-in Price. The operator can decide to display or not display some of the data by checking or unchecking the corresponding options on the top of the selection panel.

A novel feature that has been implemented in the BOS UI for the first time (to our best knowledge) is the integration of community services, which so far has only been come up as a concept in some research articles (e.g. (ACCIONA: Smart Buildings Scenario Definition)). 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

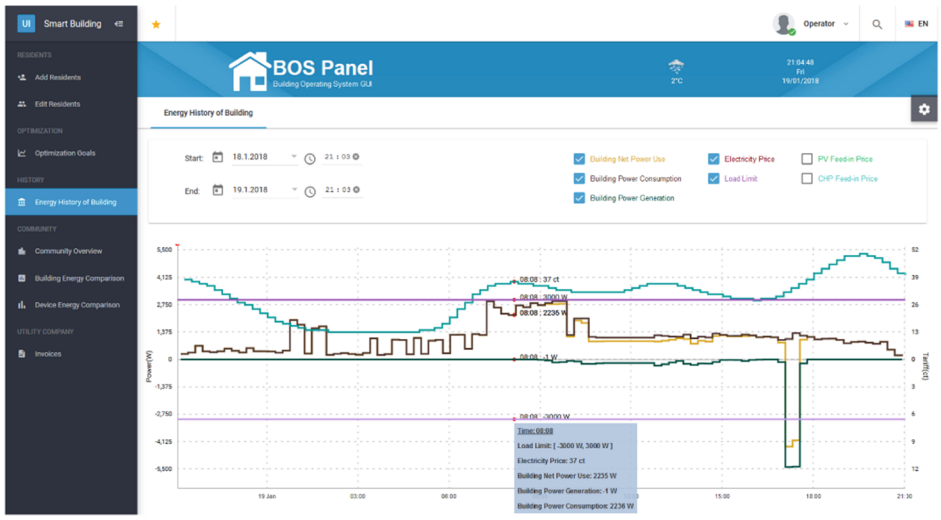


Fig. 12 BOS UI: the energy history of the ESHL

UI, the operator is able to have an overview about the communities that he has joined and not joined. The basic information about a community are predefined by the BOS UI, e.g. building types which are part of the community (residential building, office building, industrial building, or commercial building) and building numbers for each type of building in the community. Some more detailed information that might be beneficial to the operator to get a clearer view about the community could be provided by the community service provider. By joining a community, the operator is able to use the services provided by the community, e.g. comparing the energy consumption in his building with the average value in the community. To this end, he is supposed to agree to share the meter data in his building with the communities. Considering the privacy concerns, the operator can set the time granularity (15 min, 30 min, 1 h or two hours in the current version of the BOS UI) for the smart meter in the building 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.

The benefits joining a community are reflected in two aspects in the BOS UI. Firstly, the operator can compare the energy use in the building with the community for a certain month. The content of the comparison is classified into four types: Energy Consumption in the Building, Energy Generation in the Building, Net Energy Use in the Building and Greenhouse Gas Emission in the Building. Each type of the 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. Fig. 13) illustrates the average energy consumption per person (KWH/person) and the average energy cost per person (Euro/person) of the building and the counterparts in the

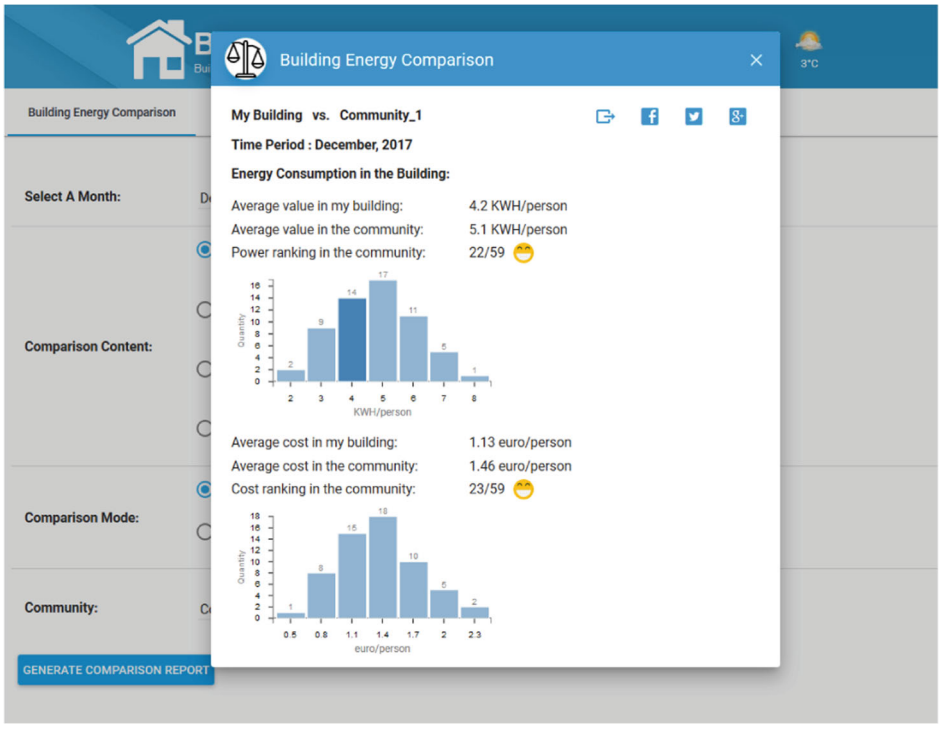


Fig. 13 BOS UI: building energy comparison in a community

community along with the building's corresponding rankings in the community. Additionally, 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 (e.g. Facebook, Twitter, etc.) by the operator.

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. 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). By utilizing the comparison capability of the BOS UI, the operator can analyze the energy efficiency of devices in his building or whether they are used by the residents in the building in a proper way when compared with the devices of the same type in a community.

Furthermore, the BOS UI is able to collect the invoices which were 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.

For the resident role, which represents the users living in the building and using the devices on a daily basis, the BOS UI provides holistic and intuitive views to visualize their energy use as well as various functionalities to facilitate advanced home automation. Figure 14 shows the energy overview page of the BOS UI for the role of resident, which consists of two parts, namely, a main panel in the middle and a side panel on the right.

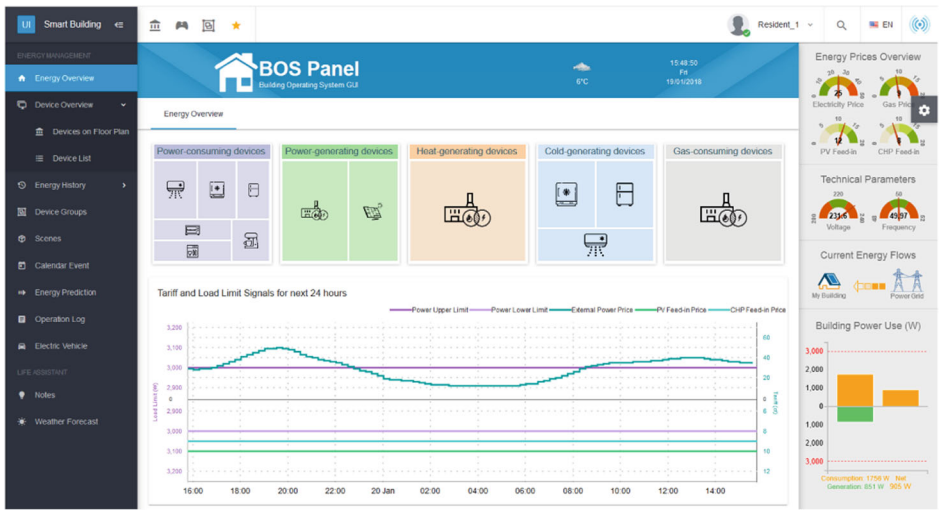


Fig. 14 BOS UI: real-time energy overview

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 the previous section. 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 information about the device (e.g. power and location,etc.) will be displayed. 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 presented 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, Fig. 15 a different layout in which the widgets have been resized and organized in a different manner, to allow for different preferences of residents.

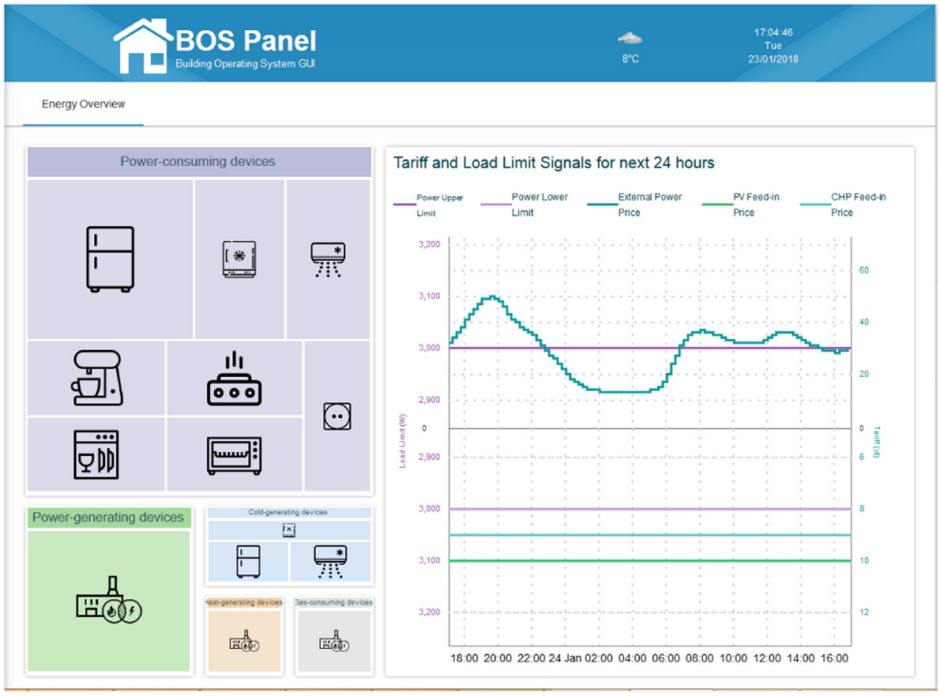


Fig. 15 BOS UI: resized and rearranged widgets for the energy overview

Next to the main panel in the middle of the energy overview page is a side panel, 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 technical parameters in the building (i.e., voltage and frequency), the current direction of the building's energy flow 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 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, 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.

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. Fig. 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. Fig. 5) described in the previous section, every 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 only a few Watts).
- Orange: the device is running and consuming power.
- Green: the device is generating electricity.

By rendering devices to different colors on the floor plan, it is supposed to be intuitive for residents to get a clear and holistic overview about the working states of the devices in their building. Additionally, the devices on the floor plan can be controlled by clicking on them. Figure 17 shows the pop-up dialog box when clicking on the washing machine on the floor plan. In addition to looking at the basic information of the washing machine, residents can also set its degree of freedom by specifying the allowed start time and required end time for their laundry. Besides, a "Turn On" button is also available in order to give residents full control to turn on the washing machine any time they want. All the information displayed in the dialog box come from the building operating system, which provides these information by instantiating the generic household device class (cf. Fig. 4). The BOS

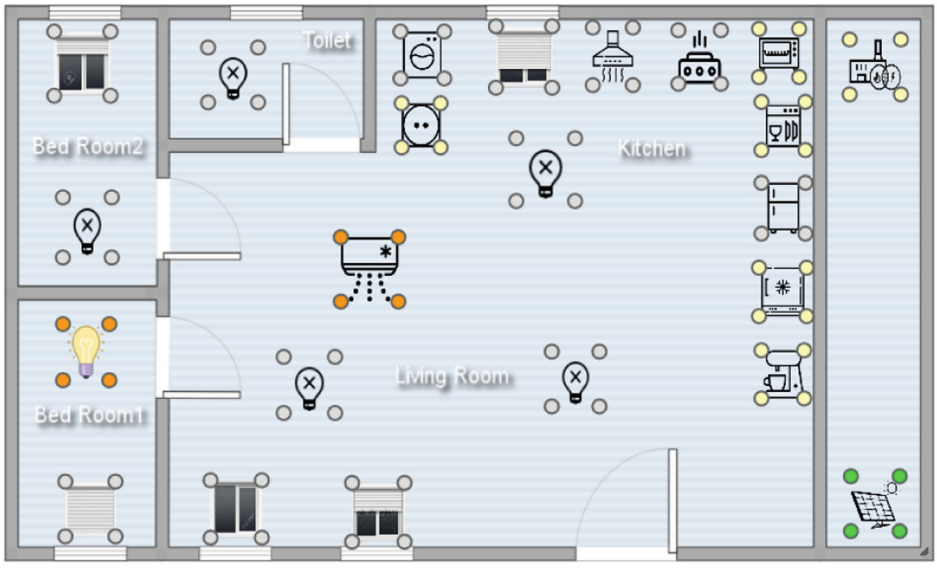


Fig. 16 BOS UI: devices overview based on a floor plan

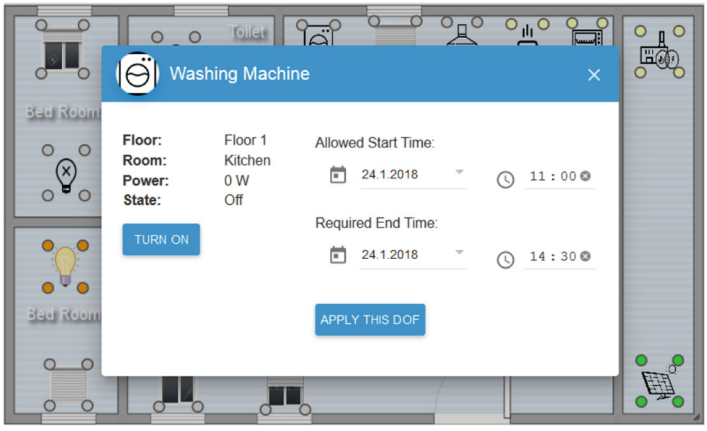


Fig. 17 BOS UI: detailed information of a washing machine

UI does not define the content of the dialog box for the device. Instead, it just traverses the device's instance object provided by the building operating system and then displays the attributes and their corresponding values to the dialog box, which ensures the generality and scalability of the user interface.

In addition to displaying devices on the floor plan, the BOS UI allows residents to view their devices in the form of a device list from different angles, e.g. locations, energy types, customized device groups etc. The various modes of viewing devices provided by the BOS UI give residents with different preferences multiple choices to get a quick overview about energy use of the devices in their building.

Home automation is the foundation of smart home/building. The BOS UI not only implements the basic automation, which facilitates residents to control devices individually, but also supports different aspects of advanced home automation, namely, device groups, scenes and calendar events. 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 a centralized control over the devices in the group. Figure 18 shows two customized device groups in the BOS UI, which contain some blinds and lights, respectively. 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.

The second way that BOS UI implements advanced home automation is based on scenes. 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 (SmartVISU a). After being created, scenes can be triggered by users whenever they need. On the BOS UI, residents are able to create different scenes according to their needs and configure corresponding target states for the devices in the scenes. Thanks to the flexible design of the household device model (cf. Fig. 4), the BOS UI is able to extract the available actions for the devices in the scene and show the widgets about the actions for residents so that they can specify the target states for the 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

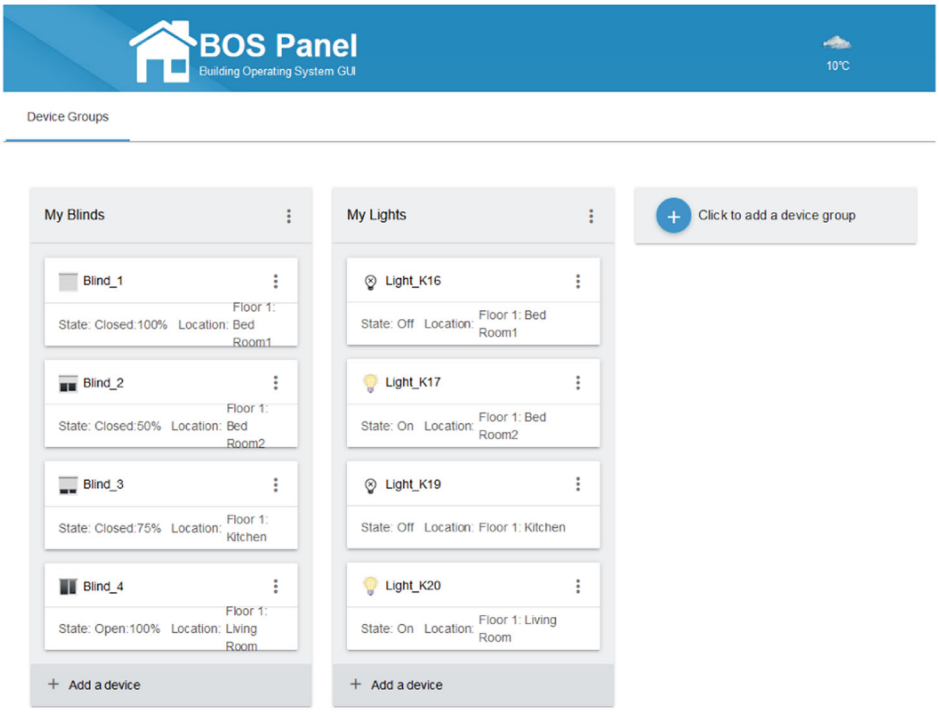


Fig. 18 BOS UI: customized device groups

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 19 illustrates 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 manipulating the widgets in the column of "Available Actions". The scenes that have been created can be triggered at any time by the resident.

Apart from device groups and scenes, the BOS UI integrates a calendar component, which provides not only basic calendar functions but also allows residents to add some events for the building to facilitate advanced home automation. The calendar events can be added by configuring devices or locations or both of them. Figure 20 shows the dialog box for adding a calendar event, which has been divided into four parts: 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 for the building, the resident firstly needs to give a title for the event and specify a start time. After that, he can choose to set locations or set devices for the event. By setting locations, he can select a location in the building where the event will take place and configure the temperature and/or humidity for the location. Besides, he can also set devices for the event by specifying target states of certain devices which are propitious to the event. After the configuration for the event start, the resident may configure states of the location and/or devices at the end of the event. Finally, the repetition of the event can be configured. One-off events, which will happen only once, can be set not to repeat. If the events are expected to be repeated, a repetition interval (e.g. every day or every week) can be set for them over a period of time.

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

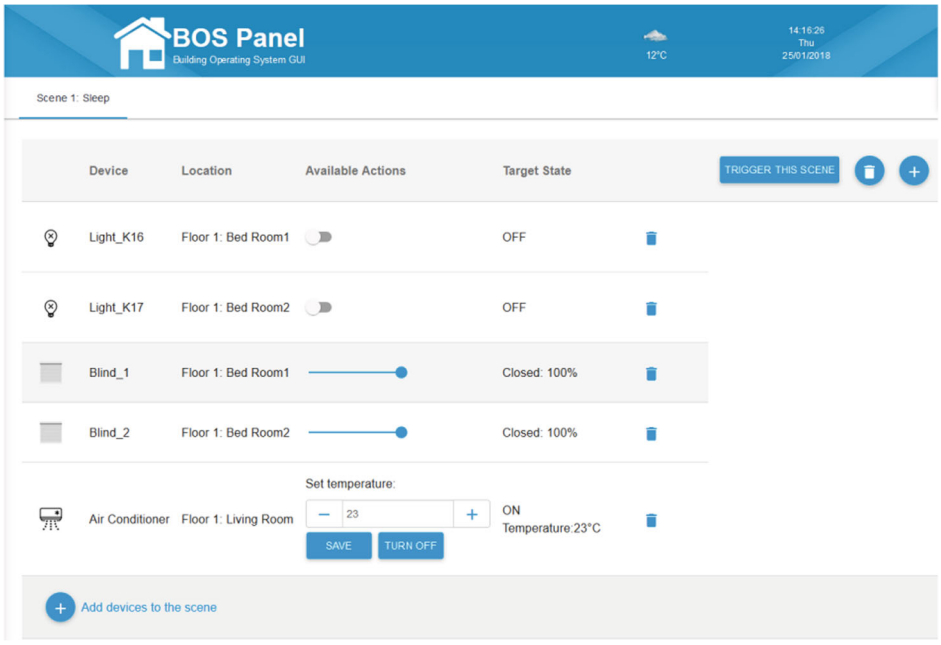


Fig. 19 BOS UI: an exemplary scene

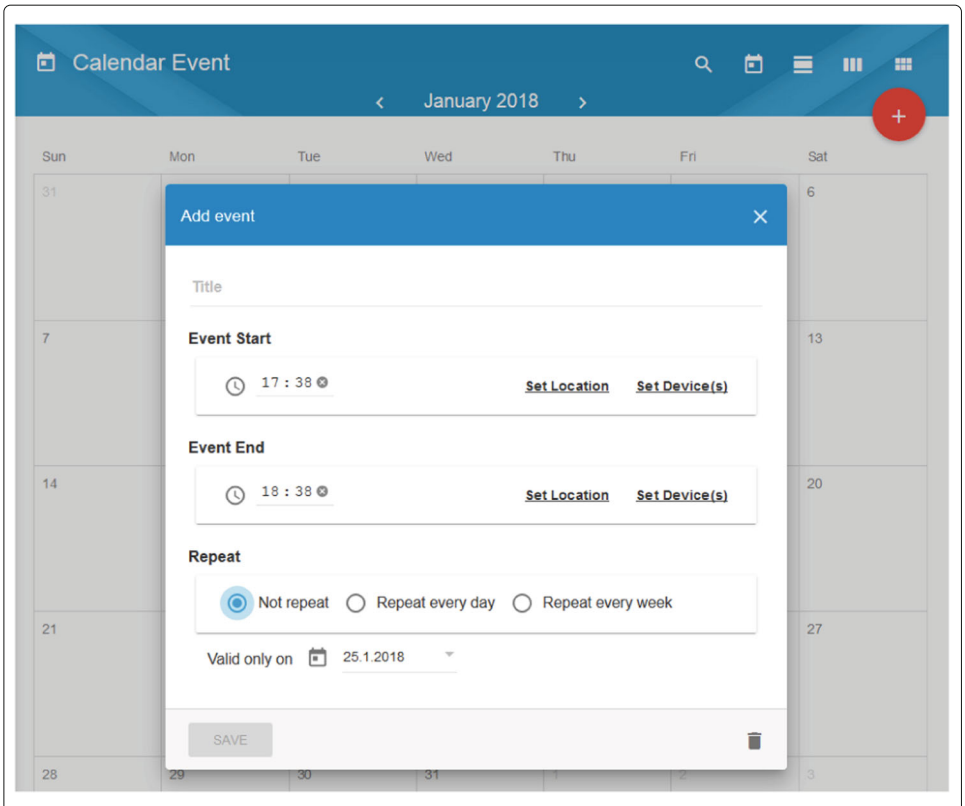


Fig. 20 BOS UI: the dialog box of adding a calendar event

Calendar event 1: Blinds automation. At 8:00 am, open all the 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 settings of temperature and humidity, switch off the lights and roll down the blinds in the room.

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. Fig. 21) is also supported by the BOS UI. With the ability of storing energy and bidirectional utilization, an electric vehicle can connect to a building and be used by building operating systems as controllable load or a mobile storage. This can 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 drive by specifying a few parameters, including the departure date and time, the distance to travel for the next trip and the minimum range that has to be guaranteed for the car to reach. The requirements specified by the resident has to be respected by the building operating system. Together with Time-of-Use tariff, load limits and the electric vehicle's maximum charging power, they determine the charging flexibility for the electric vehicle. By exploiting the flexibility of charging demand, the building operating system can 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.

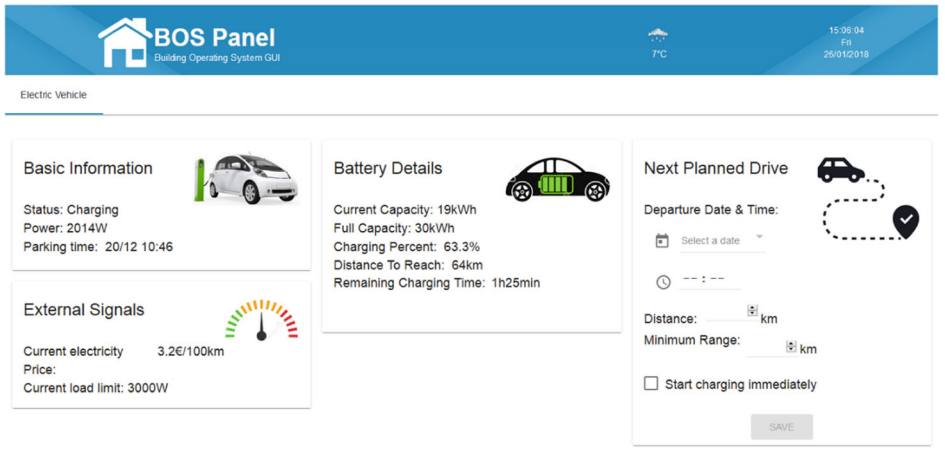


Fig. 21 BOS UI: the page of electric vehicles

Via the BOS UI, residents can not only control devices in a variety of ways but also check the energy history in their building. Since the residents have different permissions to access devices in the building, for the sake of privacy protection, the BOS UI allows residents to view the history of their personal energy use as well as the energy use history of a single device that they have permissions to use. Figure 22 illustrates a resident's historical energy use of the coffee system in the ESHL on 26.01.2018. At the top of the display is the selection panel, where the resident can choose a historical date, a device, that he is allowed 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.

In addition to checking the historical energy use of a single device, the 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 that he has permissions to use 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 the resident's overall energy use (including energy consumption/generation and corresponding cost/profit along with their average values of a community and the rankings in the community), and another part is the real-time energy use (including external load limit and energy price signals) during the selected day.

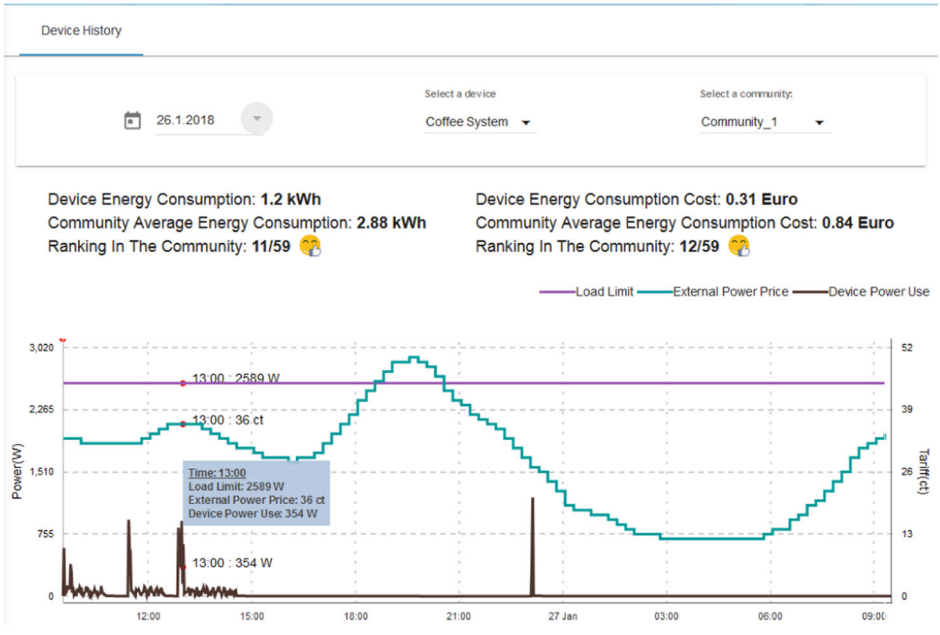


Fig. 22 BOS UI: the historical energy use of a single device

In order to provide guidance to residents for their future energy use, the BOS UI supports residents in viewing the prediction of energy use in their building as well as the external energy signals in the next 24 h (cf. Fig. 23). The available options about the predicted energy use include the base load and the net load of the building, the energy generation from the Photovoltaic and the energy generation/consumption from the μCHP. The future external energy signals displayed on the BOS UI consist of 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 Photovoltaic and the μCHP, respectively. In order to facilitate comparison, all these different types of predicted energy data and the external signals are presented 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 an operation log, which records chronological documentation of how the devices in the building are 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 of the command for the device. In the BOS UI, there are five operation modes which are defined as follows:

- **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.

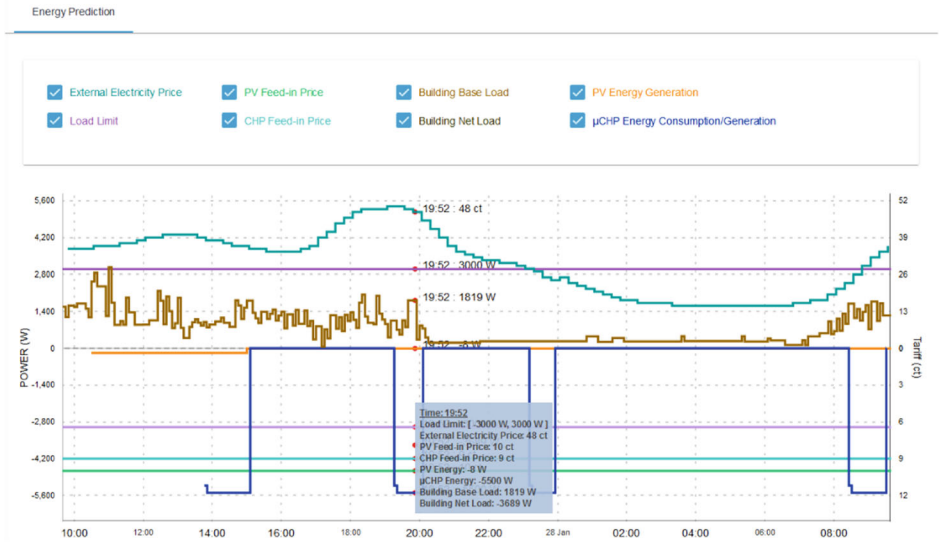


Fig. 23 BOS UI: energy prediction of the ESHL

- **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.

The BOS UI is a web-based user interface which was implemented based on AngularJS technologies, therefore, on the one hand, it can easily be accessed from any web browser by entering its URL address. On the other hand, having responsive layouts enables the BOS UI to adapt to different screen sizes. Figure 24 shows several screenshots of a tablet which was used to access the BOS UI.

In order to further improve usability, the BOS UI provides users with a number of global options which are helpful to facilitate user-friendliness. Firstly, in consideration of rich functions covered by the BOS UI, a search service is available for 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 on the top toolbar of the user interface. Secondly, benefiting from the Angular Material and the Fuse framework, both layouts and visual themes of the BOS UI are configurable. Users are allowed to choose from available layouts and color schemes to get their personalized experience. Furthermore, the BOS UI provides users with a multilingual support so that they are able to switch between different languages on-the-fly without needing to refresh the page. The function is implemented by the aid 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.

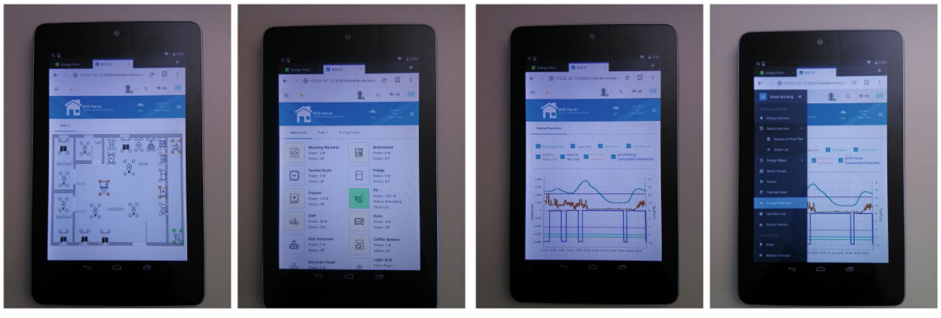


Fig. 24 The BOS UI on a tablet

Evaluation

This section evaluates the design and usability of the BOS UI by way of a theoretical and experimental analysis. 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 fill out questionnaires at the end.

Evaluation of the design

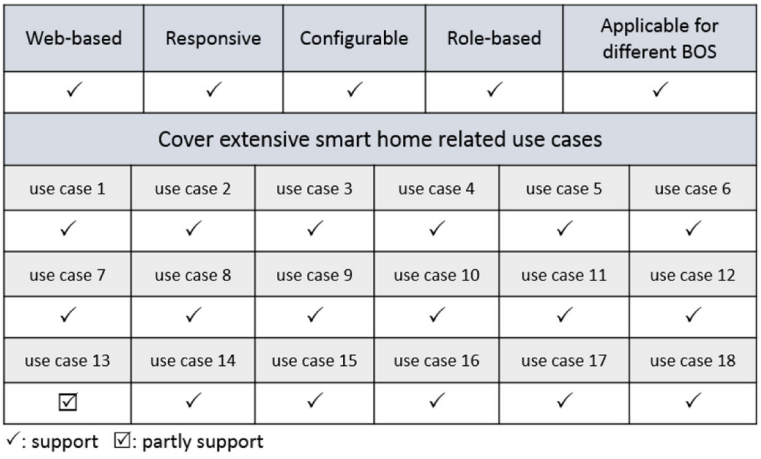
At the beginning of "Design" section, a set of requirements or criteria, that are considered necessary for any generic user interface for building operating systems, were proposed. These requirements or criteria included remote reachability, responsiveness, configurability, role management, flexibility and generality. Specifically, as described in "Design" section, we refer to a set of common smart home related use cases (cf. Table 1) in the review paper (Xu and Schmeck 2017) when discussing generality.

The aim of this section is to evaluate theoretically whether the BOS UI has met the required criteria. As the BOS UI has been specifically designed to meet these requirements, we expect them to be met to a good extent. Figure 25 shows a summary of the evaluation results, and the following are explanations of the results:

- 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 "Implementation" section, 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

Table 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



Web-based	Responsive	Configurable	Role-based	Applicable for different BOS
✓	✓	✓	✓	✓

Cover extensive smart home related use cases					
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

Fig. 25 Evaluation results for the design of the BOS UI

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 "Definitions" section, the generality of a user interface can be reflected in its support of a wide range of smart home related use cases (cf. Table 1). At this stage, all of the use cases in Table 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 article 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. 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.

The data models of BOS UI are designed in a generic way, which means they are not exclusively applicable for any of the existing building operating systems. Among all the data models of the BOS UI, the household device model (cf. Fig. 4) is the innermost and therefore the most important model, since the other data models (e.g. scene, 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 (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 data relating to the washing machine in the OSH are more than that. Other related information about the washing machine (e.g. power, degree of freedom etc.) are 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 data on the basis of its understanding to the system, consequently it is tightly coupled with the OSH. 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 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.

Table 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	infoName		infoValue		
General	Room		kitchen		
Info	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	Button	False
			eshl.miele.v1.homebus.start - 1609555631		
		Turn off	cmdString	Button	True
			eshl.miele.v1.homebus.stop - 1609555631		
DOFInfo	allowedStartTime		requiredEndTime		
	09:00, 07.02.2018		15:00, 07.02.2018		
Consumed energy	electricity				
Generated energy					

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 (OpenHAB Items). Items store different kinds of values which can be read or written, and on the other hand, they specify the way 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 3 shows the result of converting this switch Item into the household device model used by the BOS UI.

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

Table 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	cmdString	Switch	True
			1	mybroker: myhouse /bedroom /light ON		
			State	cmdString		
			0	mybroker: myhouse /bedroom /light OFF		
DOFInfo	allowedStartTime			requiredEndTime		
Consumed energy	electricity					
Generated energy						

able to utilize the BOS UI as their user interface. To this end, a middleware (cf. Fig. 26) between 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 Tables 2 and 3 is done by the

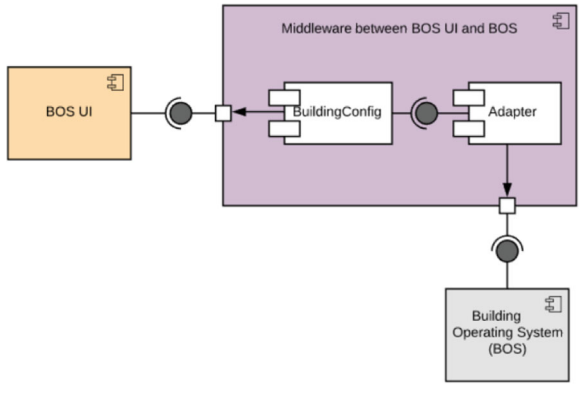


Fig. 26 The component diagram of the middleware between the BOS UI and a BOS

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.

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.

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.

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 (Bevan 2009). Usability evaluation plays an important role in the overall user interface design process since usability provides an important contribution to user experience (Hartson and Pyla 2012). The usability of the BOS UI is evaluated by testing 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 (Holzinger 2005).

More concretely, the usability evaluation of the BOS UI is done in a way of conducting experiments by inviting participants to complete pre-determined tasks and asking the participants to fill out questionnaires about the user interface. Along with the usability, a range of important functionality aspects have been evaluated using the same experiments as well. The detailed evaluation process is displayed by the flow chart in Fig. 27.

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) (Becker et al. 2012) by integrating more diversified features into it. In addition to supporting the basic functionalities provided by the EMP, the ESHL GUI is able to display

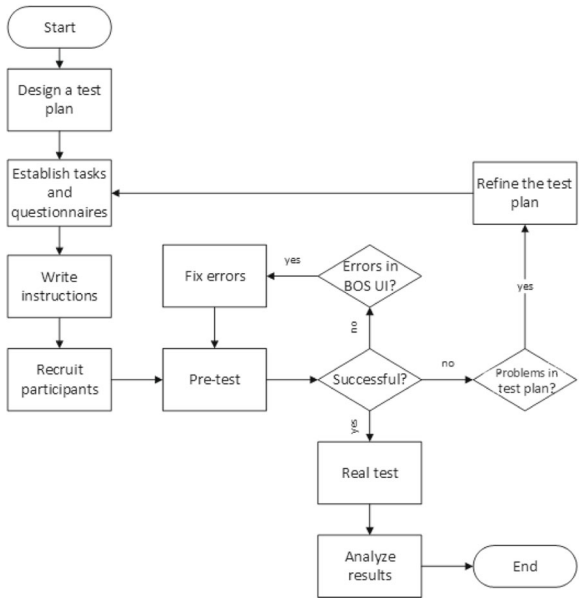


Fig. 27 The evaluation process of usability and functionality of the BOS UI

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.

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 (Dillon 2001). 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 (Bangor et al. 2008). Besides this, as Brooke put it, the SUS is particularly relevant to compare two versions of an application that are based around different technologies (Brooke 2013).

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 has a rang of 0-100 is yielded based on the answers from the respondents to the above questionnaire. The SUS score represents a composite measure of the overall usability of the system being studied. 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 (Bangor et al. 2008), which means 68 is around the 50th percentile. 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 (Bangor et al. 2009). Figure 28 shows the corresponding relations between the SUS scores, the adjective ratings, the school grading scale and the acceptability ranges.

After this theoretical preparation, a series of experiments need to be conducted in order to collect feedback of the test users on the two user interfaces. In our study, the experiments were carried out in the Karlsruhe Decision & Design Lab (KD2Lab) (KD2Lab), which is one of the largest computer-based experimental laboratories world-wide. For our experiment, the KD2Lab offers 20 soundproofed and air-conditioned computer cubicles (cf. Fig. 29). 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.

As for the participants of the experiments, the KD2Lab provides experimenters with a participants pool which has more than 2800 registered users. With the help of the 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 the experimenters to filter the users in the pool with some keywords e.g. gender, language, degree, course of studies etc., so that only the eligible users will receive 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.

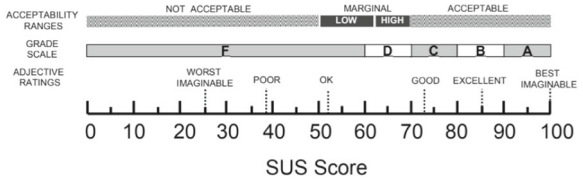


Fig. 28 A comparison of the adjective ratings, acceptability scores, and school grading scales, in relation to the average SUS score (Bangor et al. 2009)

Fig. 29 The KD2Lab

The number of participants needed for a usability test was one of the most hotly debated issues in the field (Albert and Tullis 2013). According to Nielson's article (Nielsen 2012), 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 (SAP's Article About SUS), 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 Fig. 30.

After logging in the evaluation website, the first thing that the participants need to do is to fill out a demographic survey. The purpose of this part is to collect background information (e.g. age, major subject, degree etc.) of the participants, the level of their

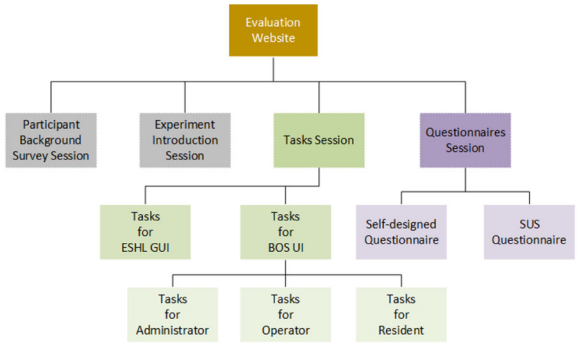


Fig. 30 The organization structure of the evaluation website

knowledge about smart home technology and their familiarity with user interfaces for smart homes, if they had ever used one before. In terms of knowledge about smart home technology, there are five levels that are available for participants to choose: no knowledge, basic knowledge, good knowledge, advanced knowledge and expert knowledge. In the actual test, the age of the 42 participants was between 18 and 40 years. The distribution of the number of participants with different knowledge levels about smart home technologies is shown in Table 4. Among the 42 participants, 4 participants have used smart home related user interfaces before. The description about the user interfaces and the participants' comments can be found in Table 5.

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 were organized according to the responsibilities of the different roles. The participants were given different roles to complete corresponding tasks on BOS UI. In total, there were 22 tasks for the BOS UI which are listed in Appendix "Tasks for the BOS UI in the experiment". 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 "Tasks for the ESHL GUI in the experiment". In order to prevent test users from having a preconceived prejudice against any of the both user interfaces, the evaluation website was programmed to display the tasks for the two user interfaces in different orders. More specifically, half of the test users started with the tasks for the BOS UI and another 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.

Forty two 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 first questionnaire are illustrated in Fig. 31. There

Table 4 Distribution of different knowledge levels about smart home technologies

Knowledge level	Number of participants
No knowledge	5
Basic knowledge	28
Good knowledge	7
Advanced knowledge	2
Expert knowledge	0

Table 5 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 app to control the light in our home.	The app 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.

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 homes.

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

Except for one test user who remained neutral, all of the test users agreed (most of them strongly agreed) 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,

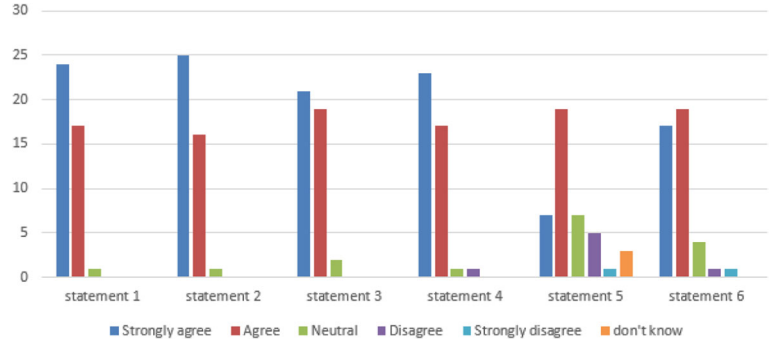


Fig. 31 The statistical results of the six statements about the BOS UI

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 Fig. 32, 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 Fig. 33. Among the total of 42 test users, 35 of them believed that the BOS UI would be able to

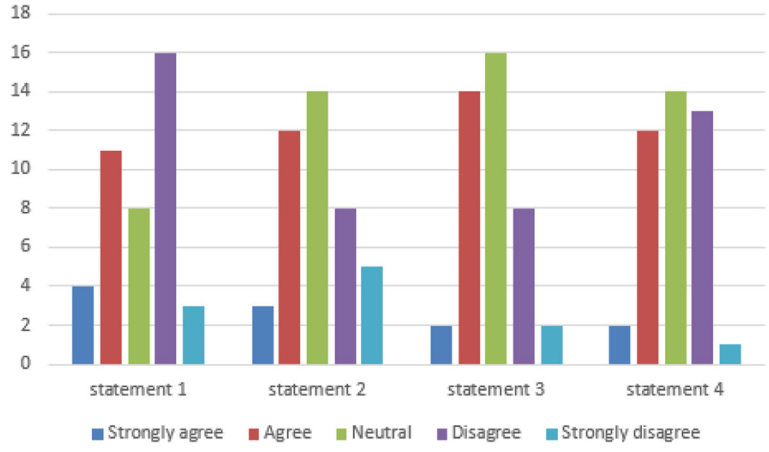


Fig. 32 The statistical results of the first four statements about the ESHL GUI

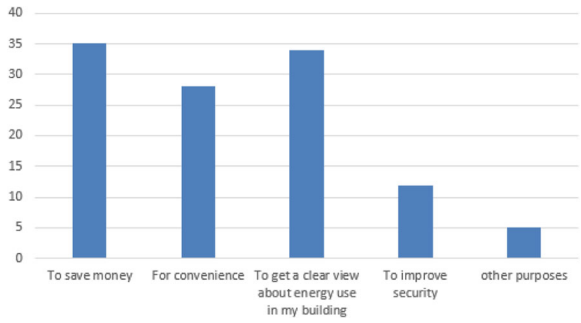


Fig. 33 Purposes of using the BOS UI as stated by the test users

help them save money and 34 of them wanted to use the BOS UI to get a clear view of their energy use in 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.

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?

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 Fig. 34, 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.

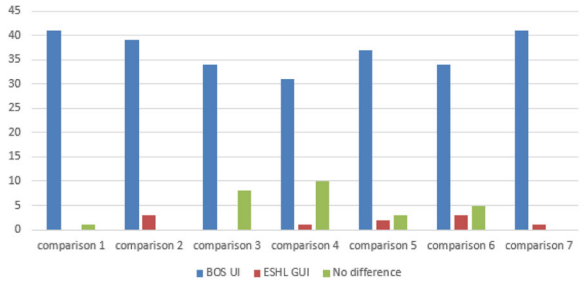


Fig. 34 The statistical results of the comparisons between the BOS UI and the ESHL GUI

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 "Comments about the BOS UI and the ESHL GUI from Test Users".

"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."

"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.

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 Fig. 28. The frequency distribution of the SUS scores of the BOS UI from the 42 test users is illustrated in Fig. 35.

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 36 shows the Normal Q-Q Plot of SUS scores of the BOS UI from the 42 test users,

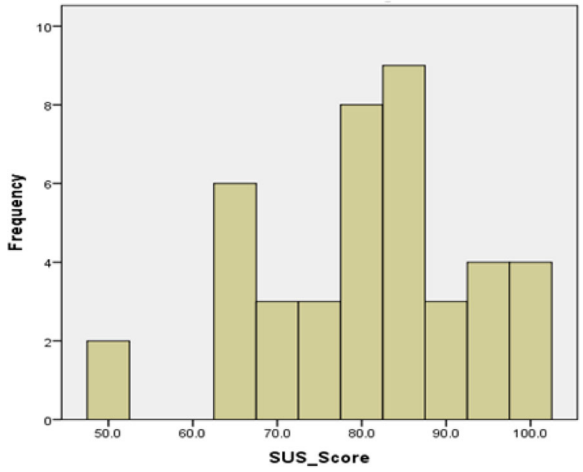


Fig. 35 Histogram of the SUS scores of the BOS UI

from which it can be seen that the points form a line that is roughly straight. The Q-Q plot in Fig. 36 provides a visual judgement that the data set conforms to a normal distribution.

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 (Ghasemi and Zahediasl 2012). 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. It is clear that both tests have a p-value greater than 0.05, which indicates normal distribution of the SUS scores.

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: $X \sim \mathcal{N}\left(\mu, \sigma^2\right)$, 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

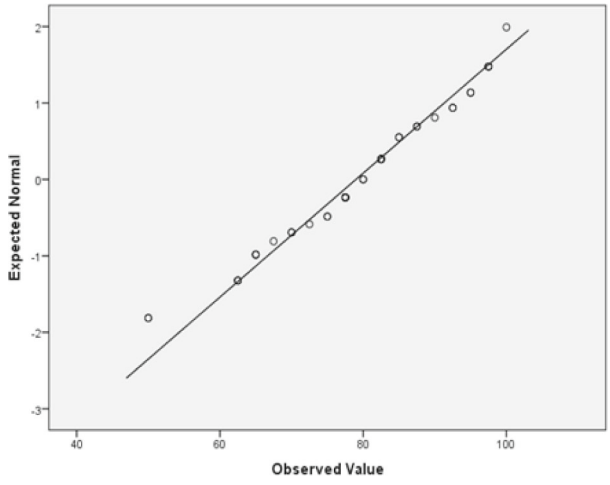


Fig. 36 Normal Q-Q Plot of the SUS scores of the BOS UI

Table 6 Tests of normality

	Kolmogorov-Smirnova[a]			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

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 (cf. Fig. 28), 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 Fig. 28, 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 Fig. 37, 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 38 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 38 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

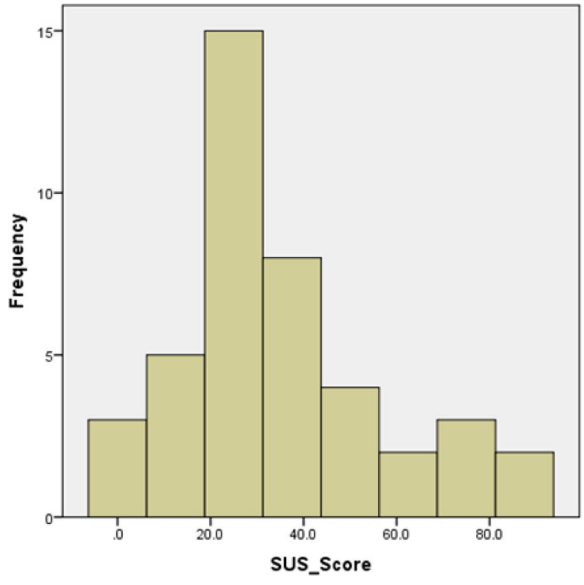


Fig. 37 Histogram of the SUS scores of the ESHL GUI

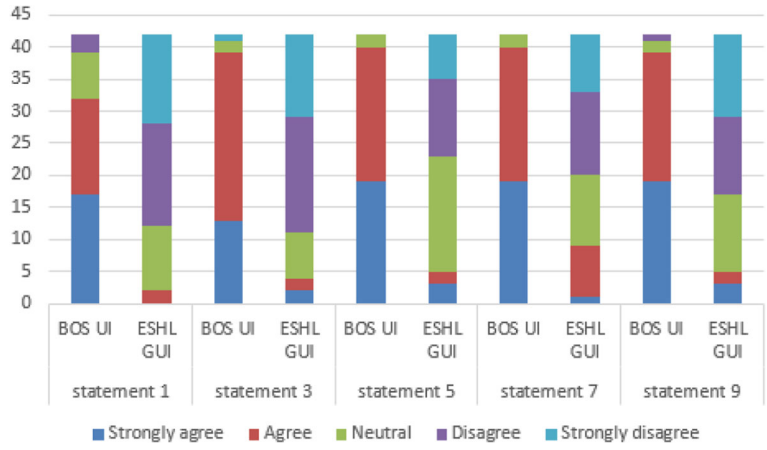


Fig. 38 Stacked bar chart of the participants' answer to the odd numbered statements in the SUS questionnaire about the BOS UI and ESHL GUI

test users thought they would not like to use it frequently. The third statement was about ease of use. Thirty nine test users, which 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 Fig. 39. Although a large number of features are integrated in the BOS

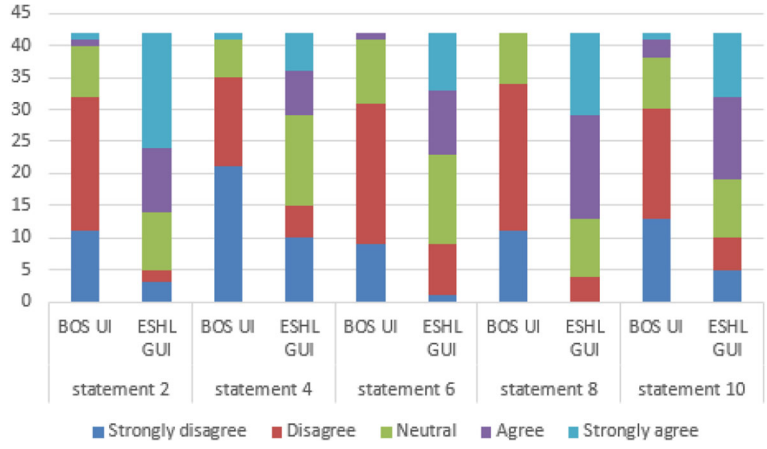


Fig. 39 Stacked bar chart of the participants' answer to the even numbered statements in the SUS questionnaire about the BOS UI and the ESHL GUI

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. Thirteen 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 40 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 4, 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 homes before the experiment. The average SUS score they

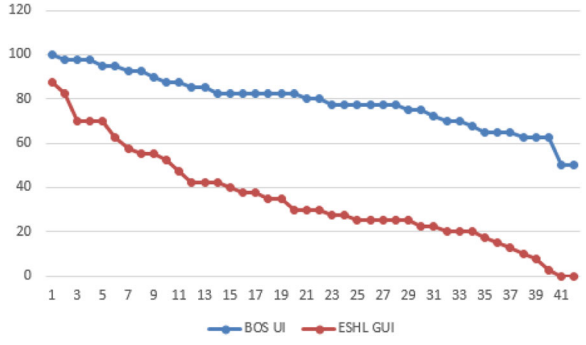


Fig. 40 The SUS scores of the BOS UI and the ESHL GUI from the 42 participants

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.

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. 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.

Discussion

Compared to the state-of-the-art and previous works in the field of data visualization and user interaction in a smart home environment, the major originality and the added-value of the study in this article is the provision of methodologies of creating a generic user interface, which can deal with different kinds of building operating systems while ensuring good usability. Specifically, the novelty or, expressed differently, the contribution can be reflected in the following aspects.

Firstly, 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 order to deal with the heterogeneity of different building operating systems, this article proposed a design, including a holistic architecture, different roles, generic data models and a number of functional components, for a generic user interface to facilitate covering hybrid energy sources in households, serving various buildings and supporting multiple advanced home automation services.

Secondly, as stated in "Introduction" section, most of the research about user interfaces for building operating systems done in the past years have dealt primarily with method description. They usually did not provide concrete user interface implementation which had already been applied to smart home instances in reality. Therefore, those user interfaces were basically either not evaluated or only evaluated based on qualitative analysis, rather than based on statistical data. By contrast, in this article, a prototype of the generic user interface, named BOS UI, was not only implemented but also applied to a real smart home environment, the ESHL at the KIT. Furthermore, in order to have a more objective evaluation of this user interface, in addition to a qualitative analysis, this article provided a quantitative evaluation to the functionalities as well as the usability of the BOS UI by inviting a group of test users to perform a number of pre-determined tasks, and then asking the test users to fill out questionnaires.

Since the ESHL, which the BOS UI has been applied to, is a smart home environment for research and demonstration, there are still some challenges, which are needed to be addressed, when it comes to implementing the design in this article in an actual household building. For instance, the building needs to be equipped with intelligent appliances, which either have smart technology built in or connect to extra smart plugs which allow remote control to those conventional home appliances, so that an IoT environment can be created. However, intelligent appliances are currently much more expensive than conventional appliances, which impedes the transition from traditional homes to smart homes.

Additionally, a building operating system is required to be installed to the household building since on the one hand, various domestic appliances from competing vendors need a unified platform to make them work together seamlessly, on the other hand, many features provided by the BOS UI, such as the realization of the specified optimization goals in a building via the user interface, need to be supported by the underlying building operating system. Besides this, a certain middleware (cf. Fig. 26) should be prepared separately in order to implement the communication between the BOS UI and the building operating system unless the building operating system is 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.

Conclusion and outlook

In this article, we proposed a concept of a generic user interface for building operating systems and a complete study including design, implementation and evaluation around this concept. We firstly proposed an architecture for a generic user interface which can deal with different building operating systems. To be able to cope with various scenarios, three roles along with their permissions were introduced. Subsequently, a series of generic data models, which were used by the generic user interface, were presented in the article. Furthermore, a prototype of the generic user interface named BOS UI has been implemented and applied to Organic Smart Home (OSH), which is a building operating system that has been deployed at the Energy Smart Home Lab (ESHL) at the Karlsruhe Institute of Technology (KIT). Finally, an evaluation combing theoretical analysis and experiments has been preformed. 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%).

Although the evaluation results of the BOS UI are positive, there are still many potential extensions to the current work that may help strengthen the functionality of the user

interface so as to expand its scope of application. For instance, the BOS UI currently does not support the configuration of its underlying building operating system due to the heterogeneity of different building operating systems, therefore an additional interface is still needed for the system configuration. To address this challenge, the potential future research may be focused on collecting and analysing 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. Besides this, some of the functions provided by the current BOS UI can also be further extended. 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. Finally, according to the feedback of the participants in the evaluation experiment, it would be favorable for the BOS UI in the future to integrate more fancy entertainment (e.g. multi-room digital music system) and security (e.g. alarms) related features.

Appendix

Tasks for the BOS UI in the experiment

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.

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

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.

Tasks for the ESHL GUI in the experiment

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

Comments about the BOS UI and the ESHL GUI from 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 GUI 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 h 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."

Acknowledgments

We gratefully acknowledge Karlsruhe Decision & Design Lab (KD2Lab), which provided experimental environment as well as test users for our evaluation experiments. We would also like to express our gratitude to Kaibin Bao, Christian Gitte, Jan Müller, and Ingo Mauser for their generous support and feedback for the work in this article.

Funding

The research in this article has been partially funded by the EIT Digital project HEGRID.

Availability of data and materials

Data sharing is not applicable to the design and implementation parts of this article as no datasets were generated or analyzed during these two parts. As to the evaluation part, all experiment sessions were recorded. The recorded videos together with experimental data sets were saved in our private repository.

Authors' contributions

HX designed the concepts of the generic user interface in this article, implemented the prototype of such a generic user interface based on the design, participated in the design and organization of the evaluation experiments, analyzed the experimental results and drafted the manuscript. LK participated in the design and implementation of the evaluation experiments, and helped to corrected and improved the manuscript. DC participated in the design of the evaluation experiments and helped to carry out the experiments. HS provided all kinds of support and feedback during the whole process of the project. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details
[1]Institute of Applied Informatics and Formal Description Methods (AIFB), Karlsruhe Institute of Technology (KIT), Kaiserstraße 89, 76133 Karlsruhe, Germany. [2]Institute of Telematics, Karlsruhe Institute of Technology (KIT), Zirkel 2, 76133 Karlsruhe, Germany.

Received: 30 July 2018 Accepted: 1 October 2018
Published online: 06 November 2018

References
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. Accessed 15 Oct 2018
Albert W, Tullis T (2013) Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. Newnes, Oxford
Allerding F, Schmeck H (2011) Organic smart home: architecture for energy management in intelligent buildings. In: Proceedings of the 2011 Workshop on Organic Computing. ACM, New York. pp 67–76
AngularJS Material. https://material.angularjs.org/latest/. Accessed 15 Oct 2018
Balaras CA, Gaglia AG, Georgopoulou E, Mirasgedis S, Sarafidis Y, Lalas DP (2007) European residential buildings and empirical assessment of the hellenic building stock, energy consumption, emissions and potential energy savings. Build Environ 42(3):1298–1314
Bangor A, Kortum PT, Miller JT (2008) An empirical evaluation of the system usability scale. Intl J Hum Comput Interact 24(6):574–594
Bangor A, Kortum P, Miller J (2009) Determining what individual sus scores mean: Adding an adjective rating scale. J Usability Stud 4(3):114–123
Becker B, Kellerer A, Schmeck H (2012) User interaction interface for energy management in smart homes. In: Proceedings of the 2012 IEEE PES Conference on Innovative Smart Grid Technologies (ISGT). IEEE Computer Society, Washington, DC. pp 1–8
Bevan N (2009) Extending quality in use to provide a framework for usability measurement. In: International Conference on Human Centered Design. Springer, Heidelberg. pp 13–22
Bladow CR, Devine CY, Schwarz E, Shamash A, Shoulberg RW, Wood JA (2000) U.S. Patent No. 6,115,040. U.S. Patent and Trademark Office, Washington, DC
Borodulkin L, Ruser H, Trankler HR (2002) 3d virtual "smart home" user interface. In: Proceedings of 2002 IEEE International Symposium on Virtual and Intelligent Measurement Systems. Institute of Electrical and Electronic Engineers, Piscataway. pp 111–115
Brooke J (2013) Sus: a retrospective. J Usability Stud 8(2):29–40
Constantine LL, Lockwood LA (2001) Structure and style in use cases for user interface design. Object modeling and user interface design. Addison-Wesley, Boston
Davidoff S, Lee MK, Yiu C, Zimmerman J, Dey AK (2006) Principles of smart home control. In: Proceedings of the International conference on ubiquitous computing. Springer, Heidelberg. pp 19–34
Dillon A (2001) The evaluation of software usability. Taylor and Francis, London
Dixon C, Mahajan R, Agarwal S, Brush AJ, Lee B, Saroiu S, Bahl P (2012) An operating system for the home. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, Berkeley. pp 25–25
EF-Pi. https://flexible-energy.eu/ef-pi/. Accessed 15 Oct 2018
FINSENY. http://www.fi-ppp-finseny.eu. Accessed 15 Oct 2018
Finster S, Baumgart I (2014) Smart-er: Peer-based privacy for smart metering. In: Proceedings of the 33rd Annual IEEE International Conference on Computer Communications (INFOCOM'14). IEEE, New York. pp 652–657
Frain B (2012) Responsive Web Design with HTML5 and CSS3. Packt Publishing Ltd, Birmingham
Fuse. http://fusetheme.com/admin-templates/angular. Accessed 15 Oct 2018
Galitz WO (2007) The Essential Guide to User Interface Design: an Introduction to GUI Design Principles and Techniques. John Wiley & Sons, Hoboken
Ghasemi A, Zahediasl S (2012) Normality tests for statistical analysis: a guide for non-statisticians. Int J Endocrinol Metab 10(2):486
Guo B (2013) Creating Personal, Social, and Urban Awareness Through Pervasive Computing. IGI global, Hershey
Harper R (2006) Inside the Smart Home. Springer, New York
Hartson R, Pyla PS (2012) The UX Book: Process and Guidelines for Ensuring a Quality User Experience. Elsevier, Amsterdam
Holzinger A (2005) Usability engineering methods for software developers. Commun ACM 48(1):71–74
KD2Lab. https://www.kd2lab.kit.edu/english/index.php. Accessed 15 Oct 2018
Latfi F, Lefebvre B, Descheneaux C (2007) Ontology-based management of the telehealth smart home, dedicated to elderly in loss of cognitive autonomy. In: Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions. Innsbruck, Austria
Macık M (2016) Automatic user interface generation. PhD thesis, Faculty of Electrical Engineering, Czech Technical University
Mauser I, Müller J, Allerding F, Schmeck H (2016) Adaptive building energy management with multiple commodities and flexible evolutionary optimization. Renew Energy 87:911–921

Molina-Markham A, Shenoy P, Fu K, Cecchet E, Irwin D (2010) Private memoirs of a smart meter. In: Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-efficiency in Building. ACM, New York. pp 61–66
Nielsen J (2012) How many test users in a usability study. Nielsen Norman Group 4(06)
OGEMA. http://www.ogema.org/. Accessed 15 Oct 2018
OpenHAB. https://www.openhab.org/. Accessed 15 Oct 2018
OpenHAB Items. https://www.openhab.org/docs/configuration/items.html. Accessed 15 Oct 2018
Orpwood R, Gibbs C, Adlam T, Faulkner R, Meegahawatte D (2005) The design of smart homes for people with dementia user interface aspects. Univ Access Inf Soc 4(2):156–164
Paetz A-G, Becker B, Fichtner W, Schmeck H, et al. (2011) Shifting electricity demand with smart home technologies–an experimental study on user acceptance. In: Proceedings of the 30th USAEE/IAEE North American Conference, Washington DC. p 20
Pérez-Lombard L, Ortiz J, Pout C (2008) A review on buildings energy consumption information. Energy Build 40(3):394–398
Roscher D, Blumendorf M, Albayrak S (2009) A meta user interface to control multimodal interaction in smart environments. In: Proceedings of the 14th International Conference on Intelligent User Interfaces. ACM, New York. pp 481–482
Sandhu RS, Coyne EJ, Feinstein HL, Youman CE (1996) Role-based access control models. Computer 29(2):38–47
SAP's Article About SUS. https://experience.sap.com/skillup/quick-ux-assessment-start-with-the-system-usability-scale/. Accessed 15 Oct 2018
Schmeck H (2005) Organic computing-a new vision for distributed embedded systems. In: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. IEEE, New York. pp 201–203
SmartVISU. https://www.eclipse.org/smarthome/documentation/features/scenes.html. Accessed 15 Oct 2018
SmartVISU. http://www.smartvisu.de/. Accessed 15 Oct 2018
The Energy Smart Home Lab at KIT. http://www.aifb.kit.edu/web/Energy_Smart_Home_Lab. Accessed 15 Oct 2018
The User Interfaces of OpenHAB 2. https://docs.openhab.org/tutorials/beginner/uis.html. Accessed 15 Oct 2018
The WAMP Protocol. https://wamp-proto.org/. Accessed 15 Oct 2018
U.S. Smartphone Use in 2015. http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015. Accessed 15 Oct 2018
Wilson C, Hargreaves T, Hauxwell-Baldwin R (2015) Smart homes and their users: a systematic analysis and key challenges. Pers Ubiquit Comput 19(2):463–476
Xu H, Schmeck H (2017) State-of-the-art user interfaces for building operating systems. In: Proceedings of the 2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC 2017. IEEE, New York. pp 283–292
Zabkowski T, Gajowniczek K (2013) Smart metering and data privacy issues. Inf Syst Manag 2(3):239–249