Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's thesis No. 64

Exploring Cloud-based Sharing of Community Recipes for Smart Environments

Dominik Olp

Course of Study: Softwaretechnik

Examiner: Prof. Dr. Albrecht Schmidt

Supervisor: M.Sc. Thomas Kubitza

Commenced: November 5, 2015

Completed: May 4, 2016

CR-Classification: H.5.2, H.4.3

Zusammenfassung

Unsere Umgebungen werden immer intelligenter. Geräte in unserem täglichen Umfeld werden durch den Fortschritt der Technik mittels Smartphones steuerbar oder nach den eigenen Ansprüchen konfigurierbar. Gibt man Nutzern die Möglichkeit das Verhalten ihrer Geräte selbst zu definieren, entsteht eine intellligente Umgebung und eine neue Art der Interaktion mit dieser. Durch die Austauschbarkeit solchen Verhaltens mit anderen, profitieren noch weitere Nutzer davon. Deshalb wurde eine Platform entwickelt, die es Nutzern ermöglicht auf einfache Weise das Verhalten einer intelligenten Umgebung mit anderen auszutauschen. Bisher ist dies ein neuer Ansatz in diesem Gebiet. Um die Auswirkungen die der Verhaltensaustausch bewirkt untersuchen zu können, wurde mittels der erstellten Platform eine Studie durchgeführt. Außerdem konnte auf diese Weise viel neue Erfahrung und Rückmeldung auf diesem Gebiet gewonnen werden. Das Erstellen der Platform erforderte weiterhin die Entwicklung eines Datentyps, der die einfache Bereitstellung sowie Verteilung von Verhalten ermöglicht.

Abstract

Our environments have become continuously smarter. With the advancement of technology, devices in our daily lives become controllable via smartphones or configurable to one's demand. Letting users define the behaviour of their devices yields a new type of interaction and creates a smart environment. Once a helpful behaviour was created, sharing such work with others, leads to the benefit of even more users. Therefore, a sharing platform was created that lets users simply create and exchange new solutions for their smart environment. To this date, the sharing of a smart environment's behaviour is a novel approach. The built sharing platform was used to research the implications of sharing behaviour between smart environments. In addition, the study was carried out to gather experience and feedback from real users. Creating such a platform required developing a data format to easily provide and distribute the necessary information to apply behaviour in another environment.

Contents

1 Introduction 9
- 1.1 Motivation . . . 9
- 1.2 Challenges . . . 10
- 1.3 Solution Approach . . . 10

2 Related Work 13
- 2.1 Background . . . 13
- 2.2 Community Aspects . . . 14
- 2.3 Smart-Systems Technology . . . 17
- 2.4 Related Platforms . . . 19
- 2.5 Security and Privacy . . . 23

3 System Concept 25
- 3.1 Taxonomy . . . 25
- 3.2 Vision . . . 26
- 3.3 Target Groups . . . 26
- 3.4 Meschup Environment . . . 27
- 3.5 Intended Functionality . . . 29
- 3.6 Comparison: App Store Concept . . . 43

4 Implementation 45
- 4.1 Meschup-Environment . . . 45
- 4.2 Technologies . . . 48
- 4.3 Structure . . . 49
- 4.4 Frontend . . . 50
- 4.5 Backend . . . 56
- 4.6 Functionality . . . 61
- 4.7 Recipe . . . 66

5 Evaluation 71
- 5.1 Evaluation Methodology . . . 71
- 5.2 Research Questions . . . 77
- 5.3 Results . . . 78

5.4 Discussion 86
5.5 Results regarding Research Questions 91

6 Conclusion and Prospect 93
6.1 Conclusion 93
6.2 Open Issues 94
6.3 Future Work 94
6.4 Prospect 97

Bibliography 99

List of Figures

2.1 Overview of Three Community Platforms 20
2.2 *Instructables* Sample Overview 21
2.3 *Instructables* Meta Information Part 22

3.1 On-Site Setup in 3D-Printed Case 28
3.2 Recipe Overview Page 31
3.3 Device Mapping Draft 36
3.4 Sample Simulator Page 40

4.1 Sample Rule on *meSchup* Server 48
4.2 System Structure 49
4.3 Detailed System Structure 50
4.4 *meSchup* Community Overview 51
4.5 *meSchup* Community Search 52
4.6 *meSchup* Community Publish 53
4.7 *meSchup* Community Recipe Detail 54
4.8 *meSchup* Server Simulation 55
4.9 URL Structure 58
4.10 Connectivity Diagram 62
4.11 Publish Recipe Sequence 64
4.12 Deploy Recipe Sequence 65
4.13 Recipe Representation 67

5.1 Hardware Utilized in Study 74

1 Introduction

The following describes the motivation behind this master thesis. Furthermore, it explains the challenges as well as the solution approach to this motivation.

1.1 Motivation

The number of smart devices in our daily environment is constantly growing. The environments we currently live in, can already be called “smart environments” since the distribution of smart consumer electronics and smart household devices is growing rapidly. On the one hand consumer electronics like smart TVs or gaming consoles no longer serve the single purpose of showing TV content, but become media centres that let users surf the internet and utilize various apps. While on the other hand basic household electronics like lamps, radiator thermostats or key locks can be bought with included wireless technology. Users are capable to interact with these types of devices for example with applications on their smartphone.
However, while the number and connectivity of such devices increases, a user’s interaction with them is still one at a time. In general, every smart device has either its own remote control or its own app to be controlled by. While there are attempts to centralize control of smart devices for example in a router (see [AVM15; Tel]) there is no industry standard that defines how smart devices can be controlled. This leaves users with solutions that can only handle a specific set of smart devices. Furthermore, if a user wants to integrate additional new devices into his smart environment, he has to wait until the provider of his smart environment supports that new technology.
The project this thesis extends, provides a solution that permits users to incorporate new devices into their smart environment. This is due to its open-source character as well as to its use of open hardware that can easily be modified and tailored to a users needs. In addition, it enables end users to exactly define the behaviour their smart environment has. The goal of this thesis is about creating a sharing space where users can publish their smart environments behaviour in order to share it with others. This way users can resort to a pool of smart environment behaviour created by others, to enrich their own smart environment.

1.2 Challenges

At the moment, a challenge is how to enable interaction between the devices in smart environments. While every device can be controlled on its own, an intelligent interaction (e.g. dimming the lights when the TV gets switched on) is not easily feasible.
Another major challenge, when implementing intelligent behaviour in a smart environment, is coping with the device heterogeneity. Since the devices in smart environments serve very different purposes, they also differ in their characteristics. A smart environment needs a method of communication that can not only tolerate different kinds of operating systems but different kinds of communication methods (WiFi, Bluetooth, Zigbee, etc.) and protocols as well.
The third challenge is enabling different kinds of users to make use of a smart environment system. While certain technology-affine users might create and tweak the behaviour of their smart environments on a hobby basis. Nevertheless, one cannot assume that every user is a technology enthusiast. Therefore, applying new behaviour to a smart environment should be easy in order to reach a huge user group.

1.3 Solution Approach

Handling the interaction of smart devices as well as the device heterogeneity is done by the *meSchup* platform [WAA+15] (see also Section 4.1). It offers a hardware solution as well as a software framework to combine various devices to create a smart environment. On the other hand, this thesis provides an approach to tackle the challenge of sharing smart environment behaviour and making it accessible for different types of users. In order to do this, a sharing platform is provided on which creators can upload their smart behaviour and make it accessible for others. Furthermore, an approach to instantly augment a users smart environment with new behaviour is implemented which makes smart behaviour accessible to a bigger user group.

Structure

The thesis is structured as follows:

Chapter 2 – Related Work: Shows the current state of development across different adjacent topics.

Chapter 3 – System Concept: A general view of the system that is created in this thesis.

Chapter 4 – Implementation: Details about the created system and functionality.

Chapter 5 – Evaluation: Evaluation details of the created system.

Chapter 6 – Conclusion and Prospect A summary of the thesis as well as a outlook of topics not covered in this work.

2 Related Work

This chapter discusses the current state of the art about topics related to this thesis.

2.1 Background

Opportunistic Design

The *meSchup* platform includes smart devices based on popular, modular hardware solutions like Arduino or Intel's Edison. Easy extensibility is one of the key features of those solutions. Based on this hardware, new types of sensors, actuators and whole devices can easily be integrated into an existing *meSchup* platform. This makes the platform very appealing to a hardware modification called "opportunistic design". It is described as: "... taking apart consumer electronics and appropriating their components for design prototypes, ..."[1] Opportunistic design takes existing, customary parts to repurpose them and build a new device out of it. Like [BB04] describes, professionals use this technique to rapidly create new design prototypes. As an inexpensive way to create new prototypes, it is ideal for users to enrich their personal smart environment. An obvious challenge of this approach is that opportunistic design naturally leads to unique devices. In order to make such devices accessible for a community of smart environment users, it must be possible to rebuild such devices. The platform introduced in this thesis provides the means to append smart behaviour with describing content about how to build and use a specific behaviour and the necessary devices.

The tradition of opportunistic design relates to user driven innovation, which is examined in [Hip05]. It is described as a process of users optimizing existing technology in order to make it more effective in solving a certain task. This user driven innovation of products is especially important in smart environment context. A user trying to optimise his smart environment has a better understanding of the requirements he puts onto

[1] cf. [HDK08] p.2

the environment than a manufacturer of smart equipment. This idea of user-centred development is picked up with this thesis in the fashion of a community based sharing mechanism. Users can modify existing solutions in order to constantly improve their smart environment. Continuing this idea by moving the implementation of modifications and optimisations of solutions away from the user is explained in section 3.5.15 on page 41.

2.2 Community Aspects

Provides an overview about community related aspects in the current research.

Programming

To successfully alter the behaviour of one's smart environment, the *meSchup* platform exploits end user programming. Since the intended purpose of this work is to enable end users to share behaviour in an online community, even less technologically aware users can access the implementations of others and modify them to suit their own needs. This method is known as "programming by example" and examined by [Nar93]. The main goal of of this approach is to reduce the complexity of programming for less experienced users by modifying existing examples to one's needs. In his work the first challenge Nardi issues is for users to find appropriate examples. In the case of this thesis the fact that a user can inspect every smart environments behaviour in order to modify it, supersedes this issue as every published behaviour would be an appropriate example. The second challenge Nardi mentions is that programming languages are not suitable for users that miss the intrinsic motivation of technology enthusiasts. This challenge is especially important in the targeted smart environment community. While the users willingly choose to enter this community, it cannot be assumed that this intrinsic motivation suffices for users to learn programming. Nevertheless, this issue is approached by a deployment strategy that lets users - unfamiliar with programming - still utilize behaviours created by others.

Additionally to opportunistic design on a hardware level, there is research regarding the characteristics of *opportunistic programming*[BGL+09]. [BGLK08] defines opportunistic programming as an activity of programming with little or no previous planning while speed has higher priority than the final robustness and maintainability of the result. One key characteristic of opportunistic programming as defined by Brandt et al. is that this

type of programming is used when building prototypes from scratch. Although creating new behaviour for the smart environment targeted in this thesis is unlike creating new application prototypes, it is assumed that similar strategies as in opportunistic programming will be taken by developers. For example another key characteristic described by Brandt et al. is the adding of new features by copying and pasting code found online. This is taken into consideration in this thesis as described in section 3.5.11 on page 37.

A research project done by Microsoft Research [BBH+14] created a cloud based IDE (called CIDE) that focuses on simple application publishing and sharing mechanisms. When creating an application in CIDE, data that is necessary to fork this application is stored. Besides the source code, information about used libraries as well as meta-data like comments, ratings or the number of downloads is stored. Furthermore, if a user forks another application in order to modify its behaviour, the information that this application is a successor will be stored. This enables the CIDE founders to track the growth and publication behaviour of applications.
With knowledge of the application source code as well as dependency tracking to find the necessary libraries, CIDE offers an easy to use sharing model. Every user can fork every published application without the need to set up complicated built environments. This greatly reduces the overhead of contributing in the community. Furthermore, Ball et al. claim that the simplicity of forking and modifying applications encourages users even more to openly exchange code. The idea of simple collaboration is the key driving force in this thesis as well.

Similar to the copy-and-paste behaviour of users described in the opportunistic programming, the research of [PBL13] went one step further. They created a plugin for Eclipse called “Seahawk”. This plugin intents to support developers by providing quick access to the popular Q& A platform stackoverflow inside of the IDE. Thereby developers are able to quickly search for solutions to programming issues without leaving the focus of their IDE. Furthermore, possible solutions appear ranked for the developer to choose. Code samples can then directly be imported into the existing source code of the developer. A persistent link to the original solution for follow-up discussions is saved as well. This form of developer support is taken into consideration in this thesis as described in section 3.5.11 on page 37.

Sociological research

Establishing an online community in the open source world, inevitably leads to the question: Why do people contribute? Developers in open source communities spend

time and effort for their work and willingly share it with the community. Lakhani and Wolf [LW03] studied this behaviour and deducted important results.
One of the key motivation factors of open source developers is intrinsic motivation based on the enjoyment of the actual task. Developers enjoy the creativity they exert while working on a project in the way they want to. The second important motivation factor is in the form of obligation. Developers, once established in the community, feel the obligation to work on community-related problems. Furthermore, Lakhani and Wolf state that this obligation is partially based on developers starting to identify themselves with the community they are affiliated with. Another important factor is extrinsic motivation in the form of payment.
Furthermore, the motivation factors were ranked according to their impact on the hours participants would work on the open source project per week. Being paid is an obvious and relevant form of extrinsic motivation, according to the study, it is a smaller driver of effort than intrinsic motivation in the form of creativity.

Although a study conducted by Deci et al. [DKR99] states that extrinsic rewards (payment) have negative influence on the intrinsic motivation of people, Lakhani et al. could not confirm this finding. This might be due to the fact that the original study of Deci was not focused in the software development environment. It is unclear how this study can be transferred to other fields. Nevertheless, Lakhani and Wolf state that they did not observe a lower intrinsic motivation in developers that were paid and feeling creative while working in an open source project.

In order to attract many people to enter a community, it is important to understand the joining process and issues that might prevent new members from joining. [KSL03] discovered four constructs that are met by newcomers when joining a community. Their research bases on an analysis of the open source Freenet project.
The first construct described is the “joining script”. It is defined as the “... level and type of activity a joiner goes through to become a member of the developer community.” [2]
Where the level of activity is described as the effort a user shows in order to become a member. The effort ranges from writing messages in mailing lists to interviews with existing developers. The type of activity on the other hand is for example to yield new feature ideas or providing real source code for a bugfix or a new feature. To the extent of this thesis, the construct of a joining script is less important, since there is no restricted access to a central code base on which members work. Instead, every member is invited to create his own solution regarding his smart environment. Nevertheless, collaboration is an important aspect of the intended community platform and enabling users to simply work together is relevant to achieve this.
The second construct is called “specialization”. It refers to a measure of how broad

[2] cf. [KSL03] p. 11

a developer contributes in the project. If somebody works on a very small subset of modules inside the project he is specialized in these, while working on a vast amount of modules means the developer is generalized. This distinction is, as the first construct, only of menial importance for this thesis since there is no classic modular system to modify.
The third construct, “contribution barrier”, describes how simple and appealing it is for newcomers to contribute to a project. It is mainly influenced by four items: The complexity of the source code as well as the difficulty to understand it; whether a future developer can choose the programming language himself to solve an issue; how easy it is to add a new module into the existing software; how independent modules are from each other. The concept of contribution barriers is very important for the intended community platform in this thesis. Trying to avoid any obstacles for users while adding new behaviour to their smart environment or publishing it at the community website is a central concept in this thesis.
The last construct is called “feature gifts”. According to von Krogh et al., a significant amount of people joined the analysed project. Their first contribution was a complete module or feature based on prior expertise in that area. Therefore, new members of a community do not necessarily start their contribution with simple bug fixing work or the like. This kind of behaviour is highly anticipated for the intended community platform in order to increase the amount of smart environment solutions available.

2.3 Smart-Systems Technology

The work by Cook and Das examines the current smart environments technology due to the increased research efforts in that field [CD07]. They state that, in general, smart devices are available even for end users, but those devices do not communicate with each other or a controller of any kind. Therefore, smart environments, in which the sensors and actuators of various devices are used, are rare. Furthermore, they conclude that until now, most smart environments only cover a part of a user’s surroundings. Users tend to have separate smart environments for work and home. Cook and Das claim further, that merging those environments to yield more intelligent environments can be the next step of research.

[KS15] created a toolkit called *meSchup*, which focuses on the rapid creation of smart environments. According to their findings, creating a network of smart devices typically leads to three important challenges.

Firstly, smart devices usually vary in the platforms they use, for example micro controllers, computers and smartphones. Therefore, in order to implement behaviour across platform boundaries, developers need knowledge to implement behaviour on all those platforms. Secondly, different devices can communicate over different channels. Computers and modern smartphones might communicate via WiFi to exchange data. However, micro controllers tend to be used in areas where no permanent energy source is available. This demands more energy efficient ways of communication like Bluetooth Low Energy or ZigBee. As a result developers need to implement ways to bridge this different communication channels. Lastly, the devices need to be deployed and connected to the infrastructure (networking, power). More about the project can be seen in section 4.1 on page 45.

Similar to the *meSchup* system used in this thesis, [DBN14] describes a centralized architecture for machine to machine communication in general. Various sensors and actuators can be connected wirelessly and communication is done via REST interfaces. To data is transmitted via the “sensor markup language”. This language consists of JSON objects that contain the data to be transmitted between the device and the centralized controller as well as additional meta-data like timestamps and device information. In case legacy devices need to be integrated that can not easily be programmed by the user, a wrapper device is used. This wrapper handles communication between the legacy device and the centralized controller. In contrast to the smart-system used in this thesis, the platform introduced by Datta et al. serve the sole purpose of transmitting data. There is no control logic available that can respond to incoming sensor values in order to trigger actuator events.

A smart-home system with a focus on energy savings was created by [NVR+11]. Again, it uses a centralized controller in order to control power outlets, smart extension cords and wireless dimmer switches. Additionally, the system can work with input elements like movement sensors. The user has control over the behaviour of the system via scripts. These scripts can utilize events such as time or movement to turn power outlets on and off or dim lights. While there is no mention of a platform upon which users could share their behaviour scripts, this work has basic support of sharing since the script files can simply be exchanged.

A commercial smart-home solution is offered by AVM, a manufacturer of networking devices [AVM15]. This solution enables users to control smart wall plugs and radiator controls in their home with their router. The wall plugs and radiator controls are connected wirelessly via the DECT standard for cordless telephone systems. Through a special web interface, schedules can be created. This is a very basic smart-home system

in which the boundaries of automation are very tight. Users can (until the date of this thesis) only control a small set of devices, which are supported by AVM and its suppliers. It does not allow for detailed programming of behaviour, which would require sensors on whose events to react on. Furthermore, it does not allow to easily integrate new devices. Users have to wait for compatible solutions until they are presented by the manufacturer.

A similar system was introduced by the German Telekom [Tel]. It is a centralized architecture that allows for the same kind of control as the solution of AVM. It supports more devices, such as fire alarms, motion detectors, sirens, lamps and others. While there are more devices to choose form than for example the AVM solution, it is again not possible to include non-proprietary devices into the system. The configuration is done via a web interface or a smartphone app. A user can configure certain "situations", which correspond to a certain set of actions (for example: lower shutter, dim lights, activate heater). This approach gives a user more opportunities of automation than the schedule approach by AVM. A sharing of predefined situation with other smart-home users is not intended.

2.4 Related Platforms

A platform with a similar idea as *meSchup* is *mbed* [ARM14]. It is based in the environment of internet of things and solely utilizes hardware based on the ARM architecture. The platform offers a specialized operating system for the ARM devices. Furthermore, software like IDEs and build tools as well as cloud services to handle device data are provided as well.
mbed focuses on building standardised commercial IoT devices. Therefore, it provides a useful set of standard software that lets developers quickly create new prototypes. Additionally, it offers software that lets ARM devices connect to the internet in order to exchange data with cloud services. This way smart devices can be created without the need to program on the low level micro controllers tend to use, or to set up a system to exchange data between multiple devices.

2.4.1 Community Platforms

Instructables [Ins14] is a closely related platform regarding its user interface and community functions. It is a website that lets users share do-it-yourself projects. There is no

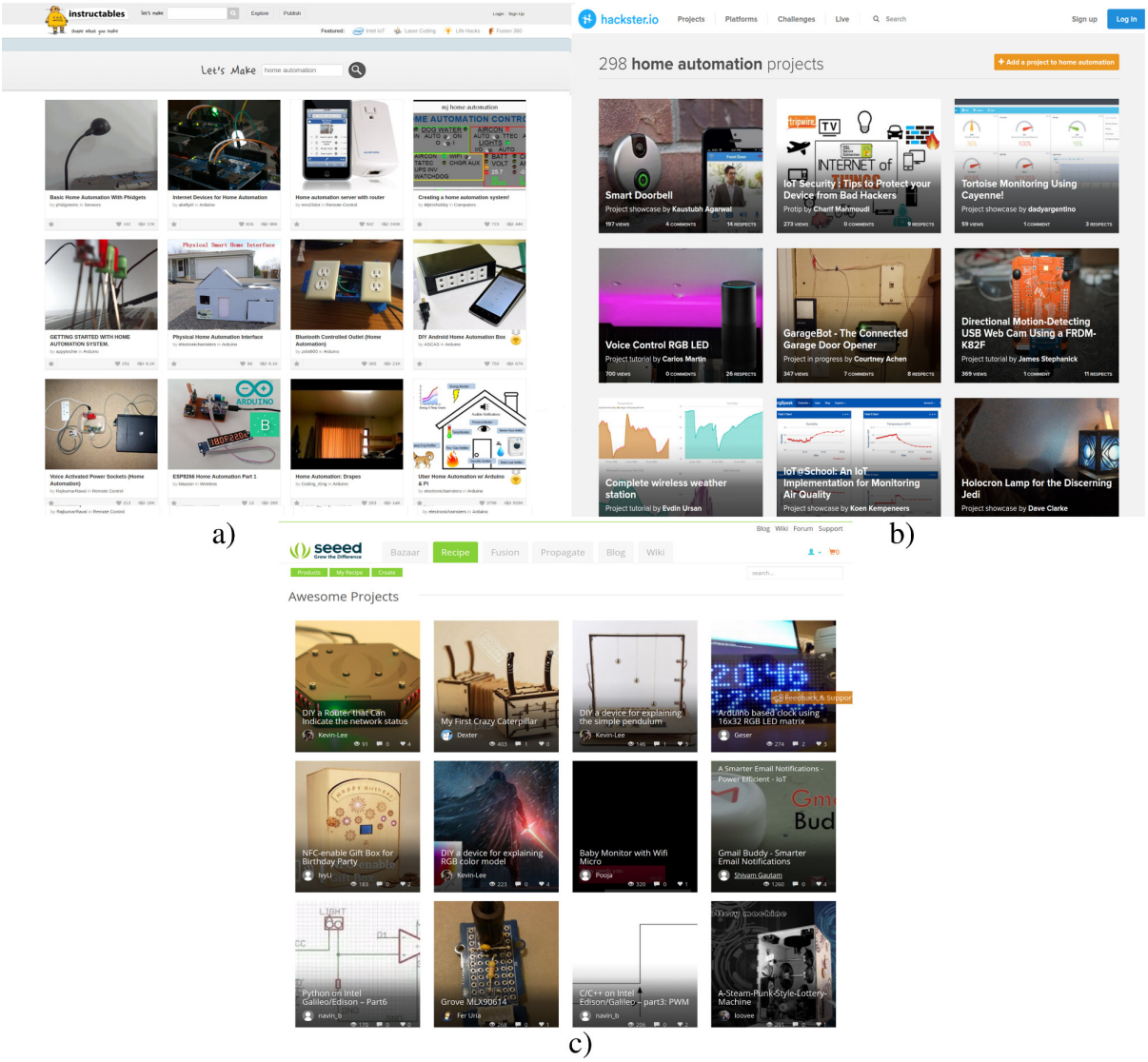

Figure 2.1: Three Community Platforms: a) Instructables, b) Hackster, c) Seeed Studio

restriction regarding the topic of a project. Users can upload instructions for electronics, cooking, tailoring and more.
It offers an easy-to-use web interface so users can quickly search for new tutorials or browse existing ones. The overview web page consists of multiple tutorials (see fig. 2.1). Each with a prominent image, the title of the recipe, the author and the category it belongs to (see fig. 2.2). Choosing any of the displayed tutorials lead to a description page. This page can contain an arbitrary amount of describing text, images or even videos. Each description page can be exported to the pdf format for users to access it offline when working on the project. The instructables community can exchange their views about each tutorial on multiple ways. Each tutorial can be commented on,

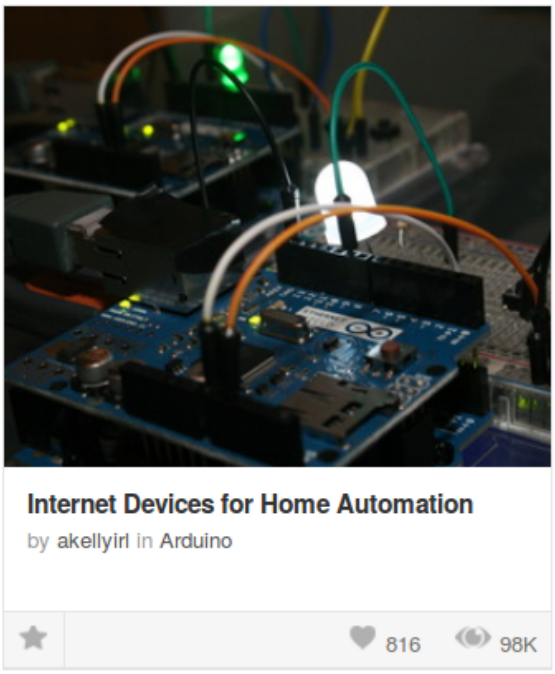

Figure 2.2: *Instructables*: Sample overview consisting of image, title, author and category

favoured or shared via social media websites. Additionally, each user can mark the projects he rebuilt. This way valuable information can be gathered about how well a project was received by the community.
In addition, the creator of a tutorial can attach his work with certain tags. These tags help users to search for tutorials as well as categorizing projects. Some of this meta information is displayed at the description page of a project (see fig. 2.3).

A similar community with a focus on electronics is *Hackster* [Hac13]. Users can share their projects with the community to get feedback or browse the selection of available projects for new ideas. Much alike *instructables*, this web site presents projects in an overview where every project is represented by a prominent image (see fig. 2.1). Attached to the image is some meta data like: Title, short description, number comments, number of views, number of likes and the name of the author.
The detailed view of every project is again structurally similar to the *instructables* website. A description of the project can be attached with pictures as well as videos. Meta information in the detailed view of the project tells the user how many other users made this project, in what category it belongs to and more.

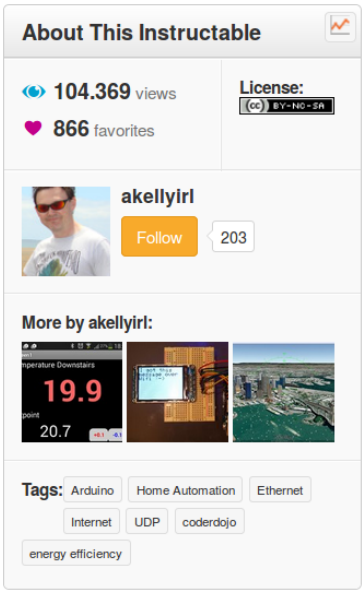

Figure 2.3: *Instructables*: Meta information part

The third closely related community platform is *Seeed Studio*[Co08] (see fig. 2.1). It is a platform focused on providing means for hardware innovations. Users can, similar to the previous communities, upload tutorials for building electronic devices of all sorts. Furthermore, the platform contains an online shop for users to buy equipment and spare parts to realise their projects. To ease implementation of a tutorial, the website shows, which items of a tutorial can be bought in the online shop. Therefore, users can browse for tutorials, order the necessary parts and see the instructions on how to implement it on the same platform.
Besides ordering prebuilt devices in the online shop, *seedstudio* offers the *Fusion* service. With the help of *Fusion*, users can order engineering samples for their projects. This includes specialized cables, circuit boards, CNC milling and laser cutting. Another service offered by the *Seeed Studio* platform is *Propagate*. This service allows users to manufacture their projects in order to sell them.
As a community platform for exchanging ideas and tutorials, *Seeed Studio* has very similar functionality as *instructables* or *Hackster*. Tutorials consist of text descriptions enriched with pictures and videos. Every tutorial can be favoured and commented on, in order to establish popularity. A feature that differentiates *Seeed Studio* from the previous platforms is a suggestion of projects with similar hardware. Next to every tutorial, users can see another set of tutorials that make use of the same or similar hardware. In that way, if a user implemented a tutorial and does no longer need the parts, he can quickly search for new projects to build, based on the same set of parts. This kind of search is an interesting approach for the *meSchup* project (see also section 6.3.7 on page 96).

2.5 Security and Privacy

With the growing propagation of smart devices secure communication and access control between devices becomes an important concern. With a focus on ubiquitous computing, [AO04] identified two main problems in creating secure smart environments. Firstly, the inherently large amount of ad hoc connections that occur between devices that had no prior knowledge of each other. Since a user might discover new smart devices at various activities (such as working, daily commute or leisure time) this is a inevitable issue.
Secondly, a solution that provides security should be compatible with the ubiquitous computing vision of actions being done in a natural, intuitive way by a user. Furthermore, a user should be able to establish secured connections between his own devices and smart devices surrounding him without the need of complex configuration tasks. According to [AO04], this problem is especially difficult since most smart devices use wireless transmission technologies like Bluetooth or WiFi "... whose integrity and confidentiality can easily be undermined by malicious entities.[3]"
The architecture developed by Argyroudis and O'Mahony solves these issues by implementing a policy mechanism into smart devices that introduces a form of access control. A smart devices does only admit access to a protected service it offers, if the requesting device can present a valid certificate. The design of the architecture lets users dynamically exchange access control policies. Due to that, users are able to quickly permit new devices access to their existing set of services offered by their devices.

Besides access control, security and privacy of data and communication is an important factor of smart environments. When arguing about security, in general three factors are addressed: confidentiality, integrity and availability. Nixon et al. summarised state of the art research regarding security and privacy (see [NWET04]). In order to introduce security in a smart environment, devices need to authenticate themselves with each other. A popular way of doing this is using a public key infrastructure (for example PGP). According to the work of Nixon et al., this is not possible in smart environments. Since PGP needs a "web of trust", which in ubiquitous smart environments cannot permanently be maintained, it is not a possible solution.
Additionally, they found out that the integrity of data is usually not sufficient in typical communication mediums. Wireless protocol implementations seem to lack resistance against attacks. Furthermore, integrity of the smart devices themselves is a substantial issue. Smart devices travel between environments while an environment has no capabilities to check, whether a device was manipulated since its last visit. In addition, since devices can be integrated invisibly in the environment, a user also lacks the capabilities to check whether a specific sensor is manipulated to present false data.

[3] cf. [AO04] p. 2

Regarding privacy in smart environments Nixon et al. refer to guidelines introduced by Langheinrich [Lan01]. These guidelines should be respected while designing and implementing a smart environment. One of these guidelines affects a users choice to interact with a smart environment. According to this guideline users should be presented with a choice whether he intents to interact with the smart environment and get the users consent if privacy related information is handled or stored. Another guideline refers to security instead. Following it means to encrypt data in a meaningful way. In smart environments it is inevitable to make use of devices that are computationally weak and therefore cannot provide strong encryption. Therefore, a sensible use of encryption and corresponding hardware is intended.
According to Nixon et al. current issues regarding security and privacy are only partially resolved and should remain a current focus of research.

3 System Concept

This chapter provides an overview of the intended smart environment as well as a comparison with existing platforms.

3.1 Taxonomy

For the remainder of this thesis, the following definitions will be used.

Smart Device A smart device is a device that includes an arbitrary number of sensors and actuators. Furthermore, it is able to transmit sensor readings and receive actuator events via a network connection to and from a central controller unit. The connection can be established both via a wire or cordless.

Smart Environment A smart environment consists of multiple smart devices. The devices within the environment produce sensor events, which trigger certain behaviour that is represented by actuator events created by the smart environment. The behaviour is fully controllable via some sort of programming by a user.

On-Site Setup The on-site setup is a single instance of a smart environment that belongs to one user. It consists of at least on *meSchup* server and any number of devices connected to it.

Rule A rule is a JavaScript based script, can be created at the web interface of a *meSchup* server. Such rules define the behaviour of the smart environment in which they are embedded.

Recipe A recipe is the combination of any number of rules attached with additional meta information. Meta information may include descriptions and images as well as information about the devices that are used in the rules. The community website handles creation, access and the sharing of recipes.

3.2 Vision

Based on the *meSchup* platform this thesis develops a system that let users share their created rules with each other. To share, users create recipes and attach them with the rules they want to publish as well as descriptive information. For this, a web based approach is planned where users can share their own solutions as well as browse solutions made by others. Ideally an active community comes into being due to the continuing exchange of ideas and recipes. In the remainder of this document the website where users can exchange those creations and ideas shall be called *community website*.

3.3 Target Groups

For the sake of readability, people who mainly *use* the platform to get recipes and deploy them on their own sever will be called *users* in this section. People who contribute and *create* new recipes will be called *creators*. Both *users* and *creators* are *members* of the community platform.

The intended target groups of the community platform stretches over multiple user groups. Firstly, the main target group are *users* that have no technical background and probably will not contribute new recipes to the community. They will make use of the deployment to get new behaviour for their smart environment and maybe provide feedback on recipes via comments or ratings. Secondly, the actual *creators* of recipes will probably be highly skilled at some technical direction of software or hardware development. They will create new rules and possibly even design extensions for *meSchup***'s** open hardware platform to be used in the community.

It is assumed that the amount of *users* will be significantly higher than the amount of *creators*. Nevertheless, in case the overall amount of *members* of the community platform is high enough, even a 95% to 5% ratio will yield enough *creators* to supply a steady stream of recipes.
Moreover, during the lifetime of the community platform the ratio of *users* to *creators* will change. At the beginning of the community more *creators* will join since only few recipes are available and *members* need to create new ones for themselves. When the selection of recipes grows, more *users* will be attracted, who benefit from the easy deployment to extend their smart environment. In the remainder of document, no distinction is made between *creators* and *users*. *Users* can advance to *creators* by learning the skills necessary

to create rules and *creators* might become mere *users* of the platform when they do no longer contribute. Therefore, both groups will simply be called *users* in the following.

3.4 Meschup Environment

The work in this thesis is based on the *meSchup* platform (see [WAA+15]). It provides means to rapidly create smart environments. This is established by the possibility for users to program the behaviour of their smart environment. Additionally, *meSchup* makes use of open hardware solutions. Thereby, users can simply create own devices and easily attach them to their smart environment.
The platform focuses on reducing the effort for end users to create behaviour in their smart environments. One of the key factors to achieve this, is a powerful abstraction of the underlying hardware and software inequalities. Naturally, in a smart environment various types of devices, each with their own operating system and communication technology, are used. In order to allow developers to quickly change the behaviour of their smart environment, an abstraction of the OS and communication technology of every device is used. This way developers can focus more on creating behaviour and less on specifics about the device they are using.
The platform is a centralized architecture and consists of a server and various smart devices that are connected via wireless technology. The server receives sensor data and can transmit data to actuators on smart devices. Creating behaviour is done by reading sensor data and initiating corresponding actuator events via a scripting language. For the remainder of this document the source code defined in the scripting language will be called *rule* The server software can be run on a regular computer or even on a less performant raspberry pi (see fig. 3.1).

3.4.1 Meschup Server

The *meSchup* server is the central controlling point of the *meSchup* platform. All sensor data gained by devices in its surroundings are combined and can be processed in the server. Every user needs one *meSchup* server at every smart location he wants to build. In the remainder of this thesis, a single smart environment (i.e. the server attached with multiple devices) is called “on-site setup”. It hosts the web interface that lets users add new devices to the system and program rules via a web IDE with the use of JavaScript. Rules can make full use of the JavaScript language to implement arbitrarily complex behaviour (see fig. 4.1 for an example).
To communicate with sensors and actuators the *meSchup* server currently supports WiFi and Bluetooth Low Energy. There are devices, which are connected to the server via

Figure 3.1: A *meSchup* server in a 3D-printed case.

WiFi and forward Bluetooth events from their surroundings. Nevertheless, devices need to be at least in the WiFi transmitting range of the server to send and receive data.

3.4.2 Sensors and Actuators

An arbitrary number of devices can connect to one server at once. A device, as defined earlier, consists of either sensors or actuators or both. Sensors deliver information of their environment via WiFi or Bluetooth to the *meSchup* server. Each transmit of such information is considered an event. While this information can represent physical readings like temperature, pressure or illumination, it is not restricted to such. A sensor event can as well be the identification string of a NFC tag in close proximity to the sensor or the gyroscope reading of a connected smartphone.
Apart from that, actuators receive events in order to change the state of their smart environment. Similar to sensors, they are not restricted to changes of the physical world around them. An actuator could be an electronic lock that opens a door when a certain device is very close. However, an actuator could as well be a display that changes its content when receiving an event.

3.5 Intended Functionality

Based on the *meSchup* platform this thesis develops a system that let users share their created rules with each other. For this, a web based approach is planned where users can share their own solutions as well as browse solutions made by others. Ideally an active community comes into being due to the continuing exchange of ideas and rules. In the remainder of this document the website where users can exchange those creations and ideas shall be called *community website*.

3.5.1 Sharing

A key factor in attracting users is making the sharing of created rules very simple. If a user decides to share rules he created in his environment the obstacle of providing them to the public should be very low. At best the user does not even have to upload any script files by himself, instead the community website automatically recognizes which rules are currently on his on-site setup and fetches them when needed. Similarly, information that might be required to configure connected devices shall be automatically retrieved as well. For more information about this data exchange please see section 3.5.10. This obviously introduces privacy issues that will be discussed in section 3.5.16.
Besides the publishing of source code, uploaded rules shall be attached with descriptive information. The descriptive information may include a title as well as a small abstract that can be used at an overview page in order to show the user a small introduction when browsing through available recipes. Additionally, it should also include the possibility to write free text enriched with images or embedded videos. The free text could be used by the developer to show what his solution does or even show example usages with images and videos. Furthermore, the description of recipe should be attachable with various documents. For example, a user might want to add datasheets regarding a device he used. For this, PDFs and regular office files should be uploadable as well.
Users should be able to create new recipes quickly. Therefore, a special view should be created where users can enter all necessary information and rules at one place. It should consist of a self-explanatory user interface and provide a preview functionality so a users can see the emerging result before actually publishing it.

3.5.2 Community Interaction

To detect the popularity of recipes in order to find out what the community wants and supports, actions should be available to each user to show his interest in certain recipes. A popular measure in this context is the rating of items. In this case, users could be able

to rate a published recipe with one to five stars. With this information very popular or very well executed recipes could be found by comparing the amount of ratings they have and the average rating value.

Besides rating there could be the possibility to comment on recipes. The amount of comments regarding a recipe could also be used as evidence of popularity. Furthermore, comments could be very useful for the publisher of a recipe to get feedback about it. With comments the community has a information channel to inform the publisher about errors or possible new features regarding his recipe. This could start an user centred innovation process where ongoing improvements of recipes are triggered by ideas from other users.

A third possibility to increase community interaction and retrieve popularity information is a method already implemented by the *instructables* and *Hackster* platforms (see section 2.4.1). It is the option for a user to mark a tutorial as done by him. In the case of the community platform intended in this thesis, a user would mark certain recipes as **in use**. This way a user looking for a certain rule in a recipe gets information about how many people are using this rule. Combined with the previous methods and based on the rating and comments of the recipe, the user is quickly able to make a judgement whether the recipe might fit his purpose.

A fourth way of creating community interaction and participation is to stage a contest. Similarly to instructables contests, the *meSchup* community could introduce contests to encourage developers to create a certain smart environment recipe. The parameters of such contests could be diverse. On the one hand, the contest could require of developers to make use of specific hardware to offer certain behaviour. On the other hand, developers might be less restricted in order to creatively develop behaviour regarding a specified topic.

3.5.3 Browsing

Similar to the platforms mentioned in the related work chapter, an overview of available recipes is necessary. It should be structured in a way that yields enough information about every single recipe that the user gets a rough understanding of what it does, yet not too much information for the user to get overwhelmed. A sensible starting point might be the representation of a recipe with an image as well as the corresponding title and a one sentence abstract of its functionality. The overview should aim to provide a

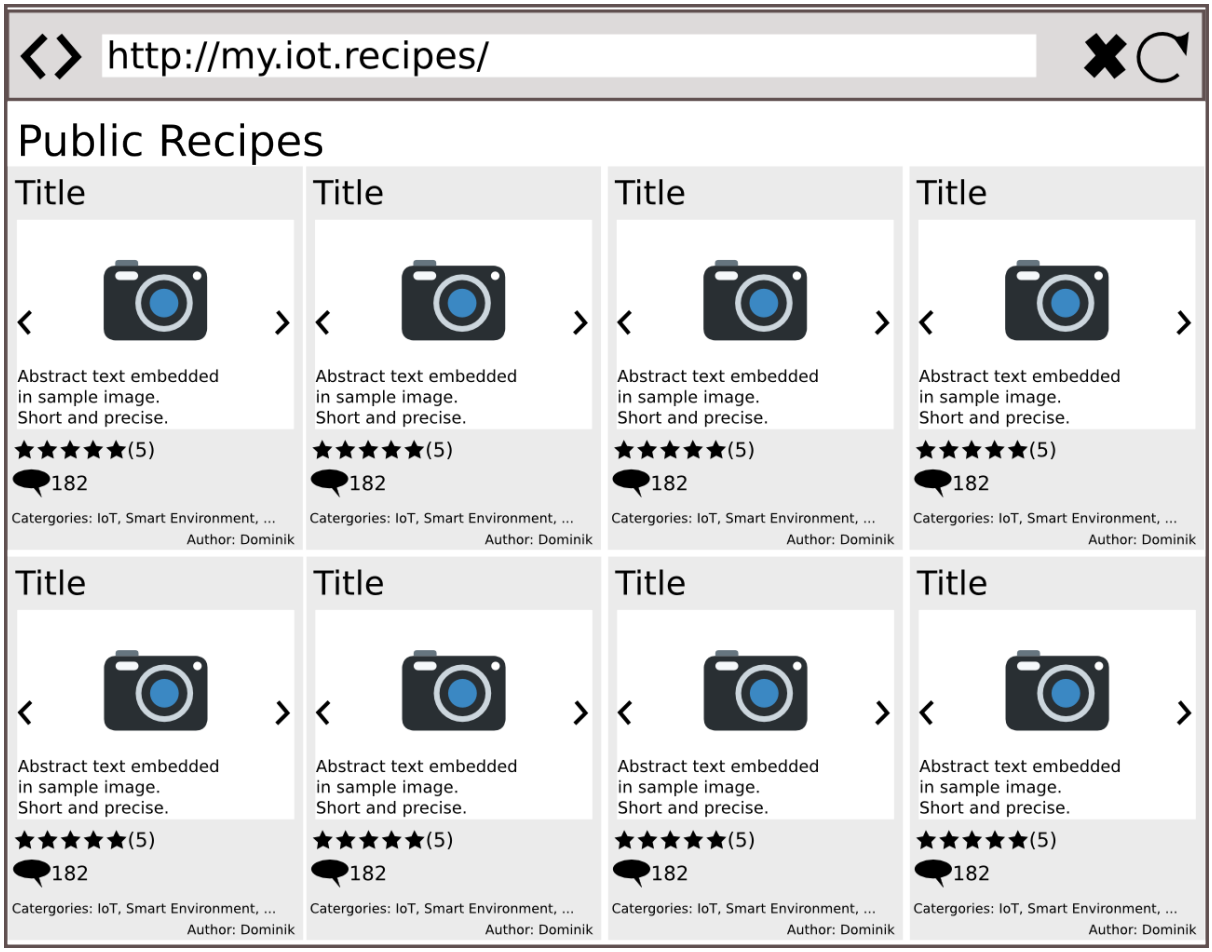

Figure 3.2: Recipe Overview Page

condensed view of a set of recipes while displaying only the most essential information. Since people tend to grasp the content of images faster than the content of texts, a layout containing a striking image seems appropriate (see fig. 3.2). The attached title and abstract of the corresponding recipe should help the user to quickly judge whether a recipe is useful for him. Additionally, every overview of a recipe should be marked according to its type. For example, a private recipe would only be visible for its creator and marked as such to be distinguishable from public ones. For more information about recipe types please see section 3.5.6.

The detailed view of a recipe should show meta information about the recipe. This could include information about the author like: who is the author or what else the author has published so far. In addition, there should be meta information about the recipe as well. This might include hardware requirements, tagging information or details about the included rule. Most importantly, the description of the recipe should be in the center

of the user's focus. It should contain all the videos and images the creator added and show them prominently to draw the users attention. The approach of this detailed view should be to show the important data of a recipe combined on one page. A user should be able to retrieve all information he needs regarding the functionality of the recipe as well as its hardware requirements and popularity quickly and effortlessly.

Besides overviews and detailed views of recipes, there should be the possibility to look into uploaded rules without previously downloading them. This could become an important feature because of two reasons: Firstly, when deploying a recipe into the own smart environment users might, naturally, be cautious. Since the deployment of a recipe could change the behaviour of the smart environment maliciously, users should be able to inspect rules beforehand. Secondly, during the development of new scripts it might be useful to inspect existing recipes as well. A developer might search for a solution of a problem in the published recipes of other users.

3.5.4 Searching

Browsing a list of recipes without any means of searching and filtering will be very difficult if the amount of recipes exceeds a certain threshold. Therefore, a powerful searching and filtering mechanism would be useful. Searching should be possible on various attributes of a recipe. The following list is not exhaustive, but indicates which part of information about a recipe should be searchable:

Title The title is the most condensed form of information regarding a recipe. The search for keywords in titles, therefore, seems very promising.

Abstract The abstract should reflect the intention and functionality of a recipe in very few words. Besides the title, it would be another promising source of keywords users may search for.

Description The description of a recipe may contain arbitrary information about a recipe. It can be an expressive source of information and due to that should be searchable by users.

Hardware A possible use case for a search might be a user looking for a recipe that utilizes his hardware equipment while providing new functionality. This use case will be possible if a list of required hardware can be created for every recipe and it is searchable by users.

Code Helping users develop new rules might be assisted by searching through existing code for examples.

Inspired by the *Seeed Studio* platform, a search based on the currently available hardware of user might bring valuable usability gains to the platform. On the basis of this search, users could extend the functionality of their smart environment without the need to add new devices. As a consequence thereof, they might prolong their membership of the community and could contribute to the liveliness of the community by rating and their feedback on certain recipes.

Both, the searching and browsing functionality are very similar to current app store concepts. App stores and the community website offer a place where all “apps” and their related meta information comes together. Therefore, users can access all available information centrally and quickly get functionality for their smart environment. More similarities and differences between the community website and current app stores can be found in section 3.6.

3.5.5 Following

As stated above, most of the members of the *meSchup* community will be *users* rather than *creators*. With this in mind, a *following* functionality would be a beneficial extension of the community website. *Following* a user would mean in this case, receiving notifications (e.g. as email) about the actions of the *followed* user. The most apparent actions being: Updating an existing recipe and creating a new recipe. Where the update of an existing recipe might range from modifying its descriptive information to modifying the rules associated with it.

This functionality is especially useful in two cases. On the one hand: Whenever a user discovers a recipe of a creator, which implements functionality, that the user wishes for. On the other hand: When such a recipe utilizes a certain set of hardware, which the user possesses. In both cases, the user could *follow* the creator to get notified about new or improved recipes.

3.5.6 Recipe Types

Different types of recipes could be beneficial for the user acceptance of the community platform. The obvious type of recipe would be *public*. This type could be downloaded and potentially be modified by every user. It is the basis the community builds on. A large amount of public recipes would help to attract users to search for their intended smart environment behaviour, download the corresponding recipe and become part of

the community. Without a wide choice of recipes, users have no incentive to use the platform or become part of it.
Another type could be a *private* recipe. It would not be visible for users that are not authorized by the original creator. This kind might be used for backup purposes if a user wants to save a copy of the rules outside of his environment. Another possible application of private recipes is as an intermediate step to deploy them on another environment. Using the deployment mechanism described in section 3.5.9 combined with private recipes, a user could transfer rules from one on-site setup to another easily with the use of the community website. For this to work, a user would generate a private recipe containing all rules needed to achieve a certain behaviour. Using the deployment mechanism, the user could then transfer the recipe to a second on-site setup without transferring files by himself or the need for the setups to be in the same network. Since the recipe could be private, even privacy related data can safely be transferred without publishing the recipe to the community.
Both types of recipes mentioned so far would aim on a single user, who created them and modifies them. For recipes to be created and modified by multiple developers, another type would be necessary. More about this kind of collaboration in section 3.5.7.

3.5.7 Access Control

To support not only the creation of recipes by individuals, but by groups of developers, some kind of access control on recipes would be needed. An initial creator of a recipe should have the option to include other developers in the *meSchup* community into a list of authorized contributors. Only those contributors shall then be able to change the content of the rules related with that recipe as well as the respective meta information. In this way it is a similar, yet greatly simplified, access control strategy as it is successfully used in popular programming communities like GitHub [Git08].

3.5.8 Revision Control

Revision control is an important tool when working with code or when collaborating with multiple people in general. The possibility to retrieve older states of a recipe for example to resolve compatibility issues should be a necessary feature of the community platform.

3.5.9 Deploying

A significant step in increasing user friendliness of the community platform is the intended *one-click-deployment*. The goal of this approach would be to enable users to integrate a recipe they found on the community website into their own smart environment with one click. For this to work there should exist some kind of connection between the on-site setup of a user and the community website. The community website would then have access to the user's on-site setup to the extent that new rules can be installed. If a user has more than one setups (for example office and home), he should be able to choose on which the recipe shall be deployed.
In order to make use of a recipe from the community website, the downloading user would need to have the same hardware equipment as the original creator used in that recipe. If he does not have the necessary hardware equipment, he should still be able to make use of the recipe through simulation (see section 3.5.14). Alternatively, in case the hardware would exist the user should select which of the available devices shall be used for the new recipe. This assignment between required devices (devices that are used in the recipe) and the devices that the deploying user possesses, would be called *device mapping*.

> **Example:** *User A creates a recipe that uses an Android device to control a wall plug. Depending on speech input on the Android device the wall plug should be switched on or off.*
> *User B has two Android devices connected in his smart environment and a wall plug. Before deployment, the community website should let user B choose which of his Android devices should be considered regarding the speech input for this recipe.*

For an exemplary draft, please see fig. 3.3. The dialogue would present the user with the possibility to go through all devices that are declared in the online recipe, and choose a corresponding device of his own on-site setup. For easier recognition, the type of each device as well as its name is displayed. This way, it is easier for the user to assign the correct devices, if multiple devices of the same type are present in the recipe. During the *device mapping*, the configurations of the deploying user's devices, are adjusted to the configurations that the creator of the recipe utilized on his devices. More information on the exchange of the configuration information can be found in the next section. This ensures a seamless commissioning of the deployed recipe. Afterwards the deployed recipe shall offer its functionality immediately.

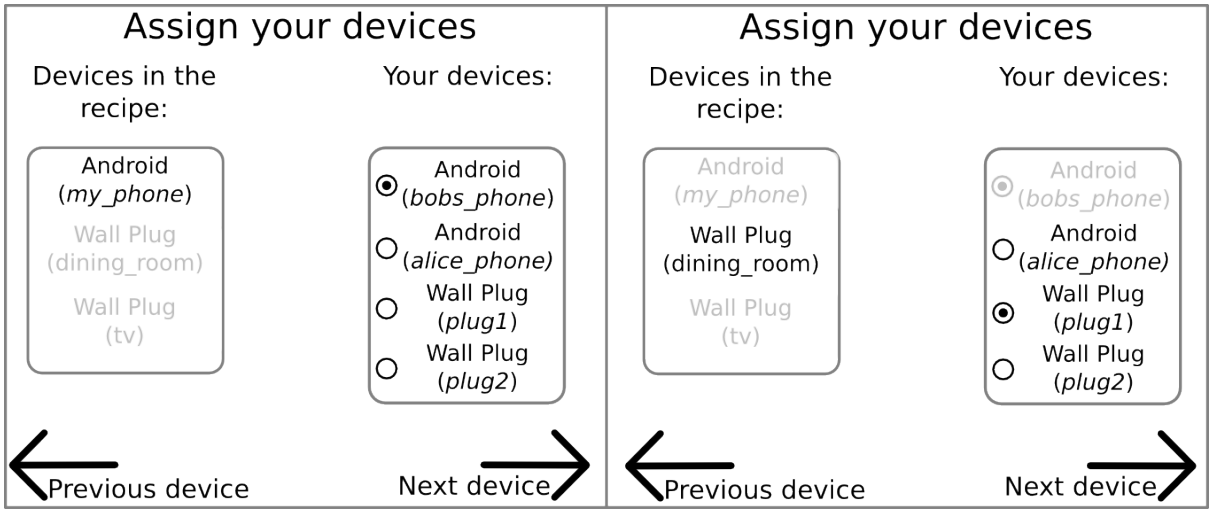

Figure 3.3: Sample dialogue for user-based device mapping. Left side: Showing the dialogue for the assignment of the first device. Right side: Same dialogue for the second device (after clicking *Next device*).

3.5.10 Data Exchange

The main information belonging to a recipe are the rules as well as the information about the required hardware for such scripts. Rules may be uploaded to a recipe via a regular file upload. This way a user could create a text file containing the code he defined at his on-site setup. This text should then be accepted by the community website when the user is going to create a new recipe. Since this requires a lot of activity on the user's side, a second approach where the data is transmitted automatically is to be preferred.
Another possibility of attaching rules to a recipe would be via a connection between the community website and a users own on-site setup. In this case the community website would be able to retrieve data from a user's smart environments. Besides rules the community website would then be able to retrieve information about what devices are used by a rule and how they are configured. Both information are vital to enable the one click deployment mentioned before. This approach has one key advantage over letting users add this data manually. Published recipes are automatically attached with correct and complete data regarding the rule and device information. Thus, it avoids errors introduced by a user when adding such information by hand to a recipe. An important issue to be considered with this approach is the intrusion into the existing on-site setup of a user. Obviously privacy has to be taken into account when using this kind of data exchange with a users equipment (see section 3.5.16).

3.5.11 Coding

Creating rules to achieve a certain kind of behaviour in ones smart environment, can be split in two areas. *End user programming* which enables users with some programming experience to create rules and have detailed control of the created behaviour. Additionally, *configuration* which adapts behaviour of rules through the adjustment of variables. Such variables represent thresholds and actuator commands and therefore change the resulting behaviour when modified.

End User Programming

The coding, or creation, of new rules should be simplified as far as possible. Implementing new scripts is done with the IDE inside the on-site setup. To enable a large group of users to implement functionality themselves, the implementation of new rules should require very little knowledge of programming. Including the community platform (the uploaded rules in particular) introduces new features that can help users to simplify and accelerate development.
Firstly, the web IDE implemented in the on-site setup could be extended by a possibility to search the online community (as described in [PBL13]). Especially the search for code or even a search in recipe descriptions or a community forum could help users troubleshooting. To further ease the search for viable solutions, query refinements as proposed by [BGL+09] could be implemented. Query refinement describes an approach to automatically attach the search query of a user with context information. In the case of developing rules this could translate to attaching the search query with information about which hardware is utilized in a certain script. For example, adding information that the user works with an android device and is utilizing the speech input to a query might lead to improved search results.
Secondly, a more experienced user might find a valuable piece of code while browsing the recipes on the community website. Instead of deploying the recipe completely on his own on-site setup, a user should have the option to select a code snippet and insert it in an rule on his own on-site setup. To distinguish the copied code from his own, it could be highlighted as discussed in [BGL+09]. Furthermore, a reference to the original recipe should be saved in order for the user to find it again when necessary. Due to the highlighting a user is automatically reminded, which code he might need to test or modify to fit his needs.

Configuration

Besides the possibility of programming behaviour, user should be able to change the behaviour implemented by a rule, through *configuration*. With this approach, the user who creates a rule could mark the conditional statements in his rule as being *configurable*. Those statements would then be parsed to retrieve hard-coded values in them.
When a user then decides to adapt the behaviour of a certain rule, he could use the *configuration* interface. The *meSchup* server would then show a dialogue, which offers the capability to change the behaviour of a rule, by changing the previously found hard-coded values with different data. Since some conditionals in a rule might break the functionality of it when modified, the user who creates a rule should be able to mark individual statements in the code. A fully automatic parsing of the code would not be able to identify such statements by itself. Upon publishing a rule, the information about which conditional should be *configurable*, would be attached to the corresponding recipe.
Configuration allows users without knowledge of programming, to change the behaviour of recipes they deployed on their on-site setups. This approach does not introduce new functionality to experienced users. Changing the source code of any rule installed on a user's on-site setup would still be possible. However, it does offer enhanced usability, especially for less experienced users, to tweak behaviour they have deployed.

Modifying hard-coded values in conditionals is the most basic type of configuration. Different approaches to automatically detect the devices used in a rule, or detect the usage of variables could be used as well. Furthermore, even the platforms capabilities of user interaction could be used to apply changes to rules. For example: The speech input capabilities of connected smartphones could be used to specify a rule via speech and, additionally, specify the conditional as well as a new value for it. This way, a user would not even need to visit the website of his on-site setup to change the behaviour of his smart environment.

3.5.12 Updates

Similar to the update functionality in common app stores, the community platform should offer some kind of notification mechanism. The notification could be done via emails that will be sent to users. Another possibility would be the usage of the already existing connection to their on-site setup. With that, a user could be notified via a pop-up message or similar when accessing the on-site's setup web page.
The kind of updates about which a user should be notified, would be restricted to rule updates, ignoring updates on the descriptive information of recipes. When the creator

of a recipe makes changes to the recipe in order to fix a known error or improve general functionality, he should be able to mark these changes as an update. Users that deployed a previous version of this recipe should then be notified that an update is available. In order to avoid security and privacy issues, updates should not be deployed automatically on a users on-site setup. Along with being notified about an update, users should be able to approve or disapprove an update. To prevent fraud, users should be able to judge whether to apply an update. The policy management system described in section 3.5.13, should assist users to make this choice.

3.5.13 Policy Management

Along with being notified about an update, users should be able to approve or disapprove an update. The actions a rule can execute should be classified in policies. To simplify this process, a system, very much alike app stores, should be incorporated. When deploying a recipe, the user will be prompted with an dialogue, asking to grant access to certain functionality for a rule. This way, the access on functionality, such as accessing sensor data, could be handled for each rule individually.

> **Example:** *User U deploys a recipe whose rule: sends HTTP requests to other websites and utilizes the sensors of an Android device.*

In this case the user would have to approve (i.e. grant access), that the rule sends HTTP data as well as allowing the rule to access sensor data generated by an Android device. Using this system, a user could prevent data being used against his will, before installing a recipe. Furthermore, in case the functionality of a rule extends through an update, the user would again be prompted to approve new policies. This way, creators of recipes cannot obtain sensitive data surreptitiously by updating an previously created recipe in a malicious manner. Describing a detailed policy management system for rules and possible use cases would extend the scope of this thesis. However, in case the userbase of the *meSchup* community is exceeding a certain threshold (and with them the need for ever improving recipes), one will have to create some kind of updating mechanism with fraud-prevention.

3.5.14 Simulating

Naturally, when a user deploys a recipe created by another both users do not necessarily have the same hardware equipment. So, a recipe created by one user might not fully work at another users smart environment if some of the hardware does not exist. To compensate possibly missing hardware a simulating feature should be available. This

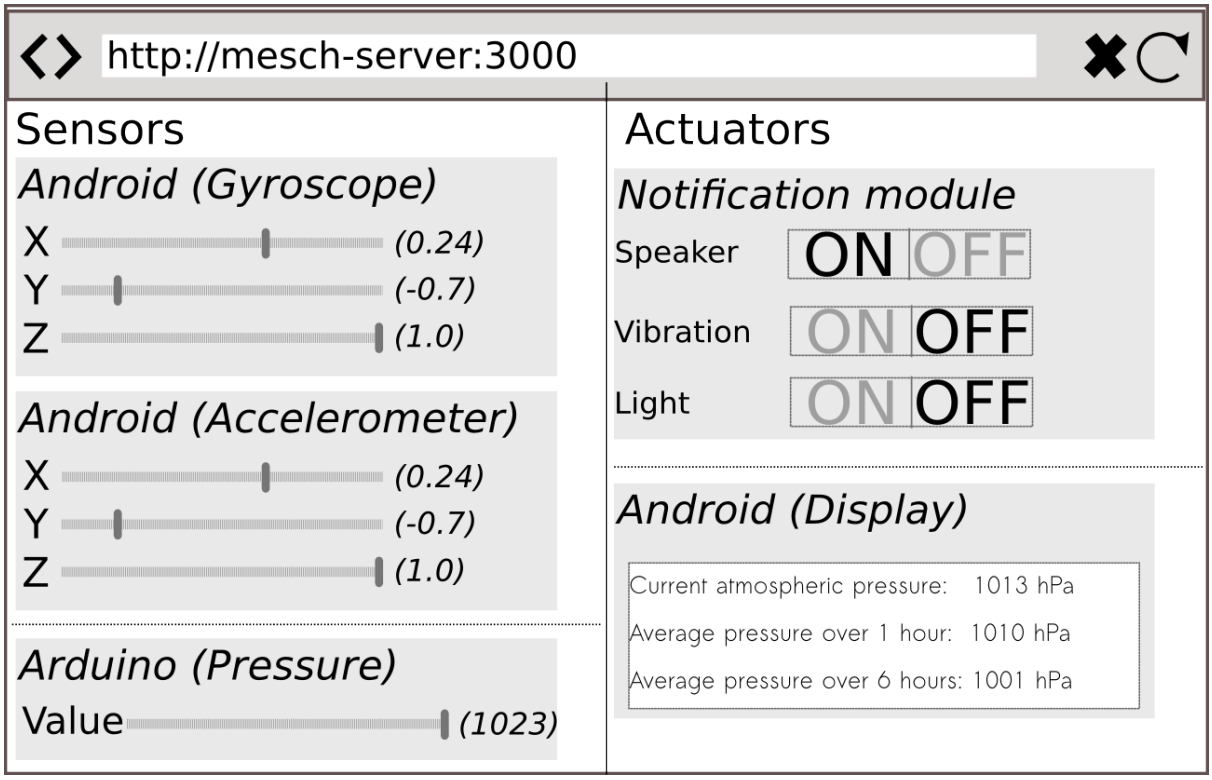

Figure 3.4: Sample simulator page. Both sensors and actors are visible and show real-time data.

feature should determine which hardware a certain rule utilizes and offer a method to simulate both sensors and actuators.
To simulate a missing sensor the feature would need a visualization of all sensors used in one rule. The user then could interact with this visualization and manually generate sensor readings, which the smart environment can react on. Actuators should as well be displayed in a visualization. If a user is missing actuators utilized in a rule he could use the visualization to see what the actuator would do. Since both sensor readings and actuator behaviour are controlled in the *meSchup* server of a user, the visualization should be integrated in the website of those servers as well (for a sample visualization, see fig. 3.4).

Besides handling missing devices when deploying a new recipe, simulating could be a valuable feature for debugging purposes. As long as a user is creating new rules for his smart environment he needs to test his work continuously. In the case of a smart environment this means triggering the right sensor events in order to check whether actuators are correctly controlled. Since the intended behaviour may be situated in the physical world generating sensor readings (i.e. changing the physical world) can become difficult.

Example: *A user wants to develop a rule that prints atmospheric pressure readings on his smartphone.*

Now to test this script the user only has the possibility to wait until the atmospheric pressure actually changes due to changing weather conditions or he somehow changes the pressure manually. Both possibilities are disproportionally time consuming. With the use of a simulating feature a user can manually generate the same sensor readings as a pressure sensor would, without the constraint to wait for the physical world to change (for a sample visualization, see fig. 3.4).

3.5.15 Monetary Aspects

With the features mentioned so far, users with little or no programming experience are at a disadvantage when they want to create rules on their own. They would need to learn the programming language and would struggle writing complex scripts. To solve this, the community platform may offer a place, in which users can state the requirements they have to a future recipe and willing developers can create those recipes in exchange for some sort of payment. Since the community platform, in its basics, is an open source platform, it is vital to keep the intrinsic motivation of contributors high. This motivation mainly originates in the creativity contributors can act out when developing new recipes. Whereas payment is a form of extrinsic motivation. In this context it is worth noting that according to Lakhani and Wolf [LW03], the payment does not have a negative effect on how much effort a contributor spends. In other words, developers will probably not start creating recipes for money instead of creating recipes for themselves and the community.

Feature Payment

One sort of payment could be a user-centred approach. In this case, a user might publish the requirements he has regarding a new recipe or regarding a single feature of a recipe. Developers could then offer their services to this user to implement the specified features. Whether this recipe would then be available for the community could be arranged by the requester and the developer.

Crowdfunding

Another way of utilizing monetary aspects for the creation of recipes would be crowd-funding. In the case of the community platform a user with an idea for a beneficial

recipe could start a crowdfunding campaign. Other users that want to support that idea to become an available recipe can join the campaign and back it with their money. The campaign starter would then need to gather money while users could offer their services as developers. Once enough money is gathered, a developer who would implement the functionality in exchange for the gathered money.

3.5.16 Security and Privacy

Security and privacy need special consideration since users share and install recipes with potentially private information in them.

> **Example 1:** *User U creates rules that control lights and radiators based on a schedule. After finishing the rules U publishes a recipe, containing the scripts, that is openly accessible.*

Others can now inspect the rules associated with this recipe and deduce, based on the schedule, when user U might be at home or on vacation. This information is private and should not be accessible by others without the original user knowing it. To prevent this information from getting published, a user should be notified when such data is found in a rule. The user should have the option to continue with the publishing or replacing sensitive information with default values. To find this kind of data, rules might need to be parsed automatically at the on-site setup before they are published.

> **Example 2:** *User U creates rules that lets him open a keyless electronic door lock when approaching the door with his smartphone.*

This example shows that not only sensitive information about the user himself, but possibly even information about his environment could accidentally be published. Depending on how the door lock works, the rule opening the door might include some kind of key that needs to be transmitted to the lock. If the rule is published with the key included, it would be easily accessible by the general public. Therefore, the lock would no longer serve its purpose since other persons could build their own “key”.

Another security issue is the direct access of the community website on a user’s on-site setup. With the one-click-deploy functionality described earlier, the community platform would have access to a user’s on-site setup and therefore his smart environment. To prevent malicious behaviour in a user’s smart environment, a deployment should only be possible by the user, who owns the target setup. Therefore, it shall be impossible for one user to deploy any recipe on another user’s smart environment.

3.6 Comparison: App Store Concept

Regarding the intended functionality the community platform is very similar to app stores in the smartphone environment. Both allow users to publish their work, get feedback from other users and simplify deployment. However, there are important differences. Firstly, an app store as it is known for providing applications for smartphones generally deploys an application for one user at one of his devices at a time. In the case of the *meSchup* platform, the deployments involve various devices. These devices need to be configured and/or simulated when applying a new recipe on a user's on-site setup. Also the behaviour that can be created, covers sensor values and actuators across different devices and platforms. Deployment in the *meSchup* environment rather equals applying behaviour than installing an application.
Secondly, the threshold to publish a recipe at the community website aims to be significantly lower than adding an application to an app store. Users should be able to quickly create and publish recipes, on the community website, as they extend their smart environment. Therefore, the main goal is to provide simple and fast tools for publishing recipes.

At the same time, there are similarities on both approaches. The community platform as well as an app store provide the means to quickly deploy recipes (respectively applications) to end users. This is important to reach users that are not familiar with programming and all the technologies behind the respective concept. Most of the users downloading applications from an app store have no technical background. Therefore, app stores have concepts to deploy applications on a user's device with very few clicks and without the need for users to have a deeper understanding of the matter. The same idea was applied on the community platform. By providing a one-click-deployment functionality, users without and technical background can use the platform to enrich their smart environment with new behaviour.

4 Implementation

In the following, implementation details about the community platform will be discussed. Whenever a users web browser is referenced, the term *client* is used.

4.1 Meschup-Environment

Section 3.4 gives an abstract overview of the capabilities of the *meSchup* platform used in this thesis. In order to explain the features of the community website in detail, the following takes a closer look at the *meSchup* platform.

4.1.1 Meschup Server

As described in section 3.4.1, the *meSchup* server is the central controlling point of each smart environment. Its main tasks are: configuring new devices that join the environment, process incoming data and execute rules.

When a user wants do add a new device to his smart environment, he can do so at the web interface of his on-site setup (his *meSchup* server). For every device, the user sees a list of ports this device has. Every port represents either a sensor (for example: accelerometer on a smartphone) or an actuator (for example: speech output). With this information, the user can choose which of the available sensors and actuators of each device, he wants to use. Using a sensor corresponds to receiving data generated by this sensor. Using an actuator on the other hand corresponds to the actuator being usable in interaction scripts.
Usually, these ports require some configuration. For example: an accelerometer can be configured to only send new data if the new data differs from the last value by a certain threshold. This way, less communication traffic is necessary since a certain sensor only sends data if there is actually new information to be sent.

Sensor data is always directed to the server and stored there. All this data is stored in a way, that it can be accessed by rules. This allows users to program behaviour that can interpret sensor data and to correspondingly react on it via actuators.
Furthermore, every time sensor data is received by the server, all available rules are executed. This way the user does not have to specify at which point in time a recipe should run, but can rely on it being run as long as sensor data arrives.

All communication between the *meSchup* server and the connected devices is done via wireless transmission. Currently, the server can communicate via WiFi, Bluetooth, XBee and RF Link. The integration of the server in a users home network can be either WiFi or regular ethernet.

4.1.2 Sensors and Actuators

Sensors and actuators of the smart environment can be various devices. Besides the relatively expensive android-based devices, *meSchup* also supports hardware based on *Arduino*, *Intel Edison*, *Gadgeteer* and *Raspberry Pi*. For all those devices there is a customized firmware available, that can be installed on the devices directly from the *meSchup* website. When installed, it allows to easily embed these in the *meSchup* platform. This type of device usually needs additional sensors or actuators in order to provide some kind of functionality. In general, these devices offer ports, where such equipment can be connected. Nevertheless, complex functionality usually needs to be built by the user himself.
Besides, there are pre-built devices that are supported by the *meSchup* platform. These devices have sensors or actuators included and work instantly or after a firmware change. For example: a smart wall plug is available that can meter the power consumption of a plugged in device as well as switch it on or off.
For a device to be recognized in the smart environment, the user has to configure and prepare each device in his on-site setup.

4.1.3 Programming

The programming of interaction scripts utilizes powerful abstractions. As mentioned earlier, various devices, each with different communication technologies, might be used in the same environment. To alleviate the user of handling all the differences that emerge when working with different operating systems and transmission protocols, a high-level programming approach was chosen. Due to that, a user can access sensors

and actuators of devices with one line of code and without knowledge about the intermediate communication technologies. With the current state of the *meSchup* platform, the implementation of basic behaviour does require very little knowledge of programming. Nevertheless, arbitrarily complex rules can be created, utilizing the full power of the corresponding programming language.

All the programming is done with the widely popular JavaScript language. It is easily readable and a powerful object oriented programming language that assists users in creating behaviour quickly. Furthermore, an auto-completion feature is implemented at the *meSchup* user interface to make development even more rapid and less error prone.

The programming methodology applied is event-based. This means that every sensor reading sent to the central server, as well as every actuator message sent to a device is considered an event. For every incoming sensor event at the server, the ruleengine is triggered. The ruleengine itself handles the execution of every rule saved on the server. Every time the engine is triggered, all available rules will be executed, irrespective of whether the incoming sensor data is actually used in a rule.

Rules do not need to provide functionality in terms of creating actuator events. A rule can also solely consist of utility functions, that can be utilized in other rules. Furthermore, rules can be combined to groups. That way, a rule which only provides utility functions to make another rule more readable, can be combined with this rule into a group.

Priorities, that influence the order of execution when the ruleengine is triggered, can be set to both, rules and groups. This is useful especially in the case of utility functions as described before. In JavaScript, functions need to be declared before they can be used, therefore the order of execution of scripts is important. It is necessary to execute the rule with the definition of utility functions before their actual usage.

In fig. 4.1, a sample rule is shown. It solves the following task:

> **Task:** *A display connected to a Raspberry Pi shall show a message about whether company is desired, depending on the position of the user's smartphone. When the display faces downwards (e.g. the desk), a "do not disturb" message shall be visible.*

```
if (api.device.AndroidDevice.GyroscopeOne.Z  < -0.8) {
    api.device.RaspberryPi.WebDisplayOne.showText = "Please do not disturb."
} else {
    api.device.RaspberryPi.WebDisplayOne.showText = "Please come in.";
}
```

Figure 4.1: A sample rule displaying a text message on an external display.

4.2 Technologies

Various technologies were used during the development of the community platform. The most important ones will be discussed in the following.

4.2.1 Database

CouchDB is used as a database for recipe related information (see [Apa05]). All data that is necessary to deploy and visualize a recipe at the community platform is saved in a *CouchDB* instance, with the exception of images. *CouchDB* was chosen because of various reasons that will be discussed further in section 4.5.1.

4.2.2 User Interface

For the user interface, the CSS framework *Semantic UI* is used (see [Luk13]). Using *Semantic UI* it is possible to develop HTML and JQuery based websites with a responsive design very quickly. It offers a huge amount of HTML elements that can be embedded in a website as well as various modern themes.
Additionally *JADE* (see [Lin10]) is used as a template engine. Template engines allow developers to write condensed, easily readable code that will automatically be translated into a more verbose and less readable language. In the case of *JADE*, it allows to structure websites based on a set of keywords that are automatically translated into valid HTML. It can be combined with *Semantic UI* to create maintainable code while utilizing the advantages of *Semantic UI* in creating appealing interfaces.

4.2.3 Backend

The backend server utilizes *Node.js* as a runtime environment (see [Joy09]). It's a fast, event-driven runtime environment that makes use of asynchronous I/O operations. This

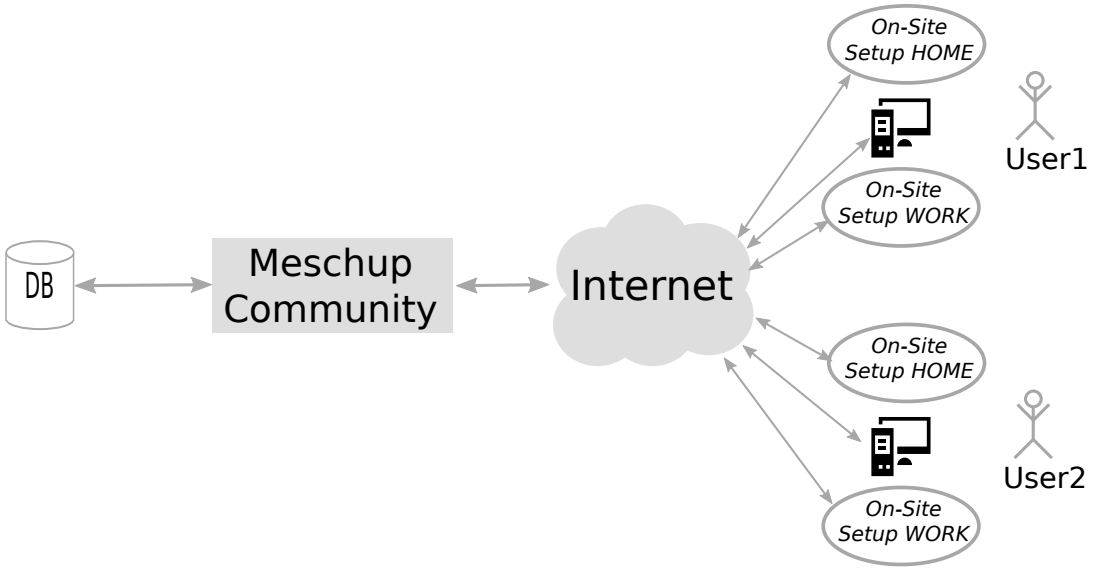

Figure 4.2: The abstract structure of the *meSchup* platform and the community server.

means the runtime does not halt and wait for operations, such as a web request or disk access, to finish. During the time of an ongoing I/O operation, the application continues with the execution of subsequent code. This is an essential factor to create applications, that are both scalable and responsive to end users.

4.3 Structure

An abstract view of the *meSchup* community server as well as the setups of end users can be seen in fig. 4.2. The system consists of two main servers. On the one hand there is the meschup community application that is hosted on one of *Amazon's EC2* instances. This was done to be able to quickly scale up the application, in case the amount of users grows and the load on a single server would be too high. On the other hand, there is the *CouchDB* application that runs on a single v-server. Clients as well as on-site setups are connected to the community server via the internet and do not directly target the database server.

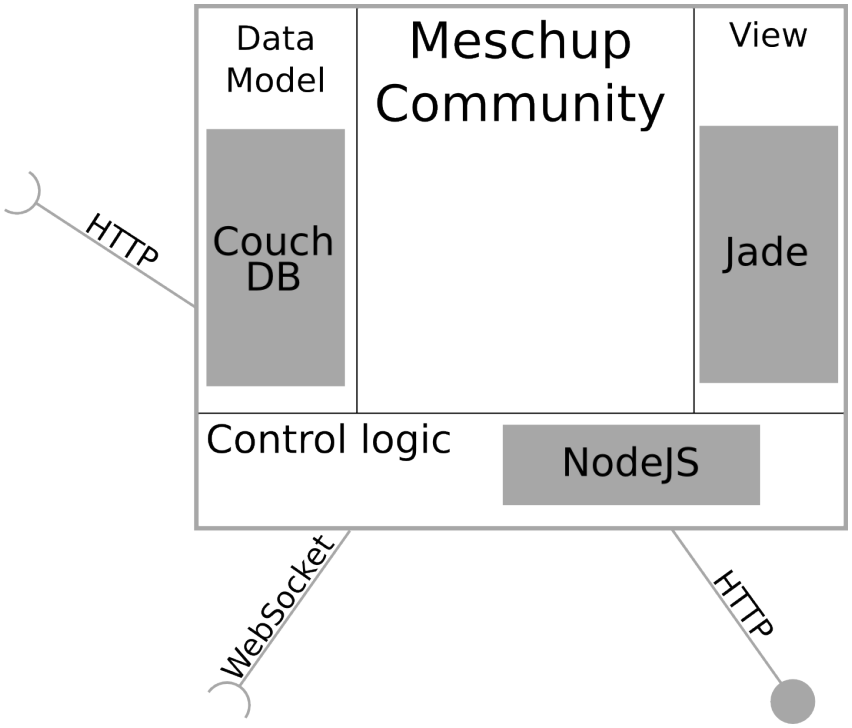

Figure 4.3: A more detailed structure of the community server. A filled circle represents an API provided by the community server, a half circle represents the usage of another API.

A more detailed view of the community server is shown in fig. 4.3. The server offers a HTTP interface that is used by clients (i.e. web browsers). Besides that, it is depending on two remote interfaces. Firstly, the HTTP interface of the database as given by *CouchDB*. Secondly, the interface of the on-site setups which is accessible via a *WebSocket* connection. Further information about these connections can be found at section 4.6.1.

4.4 Frontend

This section gives an overview of the features accessible through the web interface of the community website

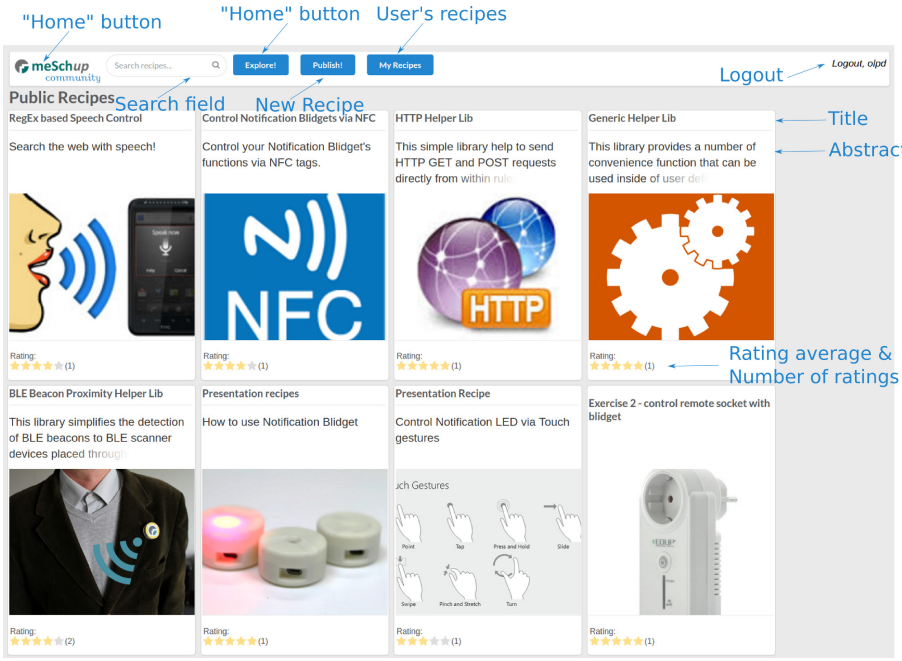

Figure 4.4: Overview page with small representations of all available recipes.

4.4.1 Overview

Figure 4.4 shows the overview page of the *meSchup* community platform. The top banner is consistent over all views of the community website, except the code view. It works as a navigation bar and allows the user to navigate quickly to any page. It consists of multiple buttons. Starting from the left:

Icon The logo works as a “home button”. It shows the *meSchup* icon and will relocate the user back to the overview from any previous page.

Search Field The search field allows a user to search the available recipes, a detailed description follows in section 4.4.2.

Explore Button The *Explore!* button has the same functionality as the community icon.

Publish Button After that, the *Publish!* button is located. It will relocate the user to another page where he can create new recipes and upload rules from his on-site setup. More information will follow in section 4.4.3.

My Recipes Button If clicked, it will redirect the user to a identical looking overview page, displaying only recipes created by him.

Logout Link At the far right corner of the banner, there is a link to end the current session, named “logout”. If no user is logged in, it instead shows two links to either login or sign up for a new account.

If no user is logged in, the overview shows all publicly available recipes. A logged in user sees all publicly available recipes as well as his own private recipes. The overview page is the starting point of the community website. Pressing one of the “home” buttons (the logo in the top left corner or the “Explore!” button) will redirect the user back to the overview, independent form the previous location.

4.4.2 Searching

Searching is done via the search field in the top banner. A search query is started, every time a user enters a character. Due to that, searches on partial words are possible. Results of the search are displayed in a box right below the search field (see fig. 4.5). Every recipe that matches the user’s search query, is displayed as one line containing the title and abstract of the recipe. At the bottom of the results box is a link, which redirects the user to a search page. The search page has the same layout as the overview page, and shows all recipes matching the search query.

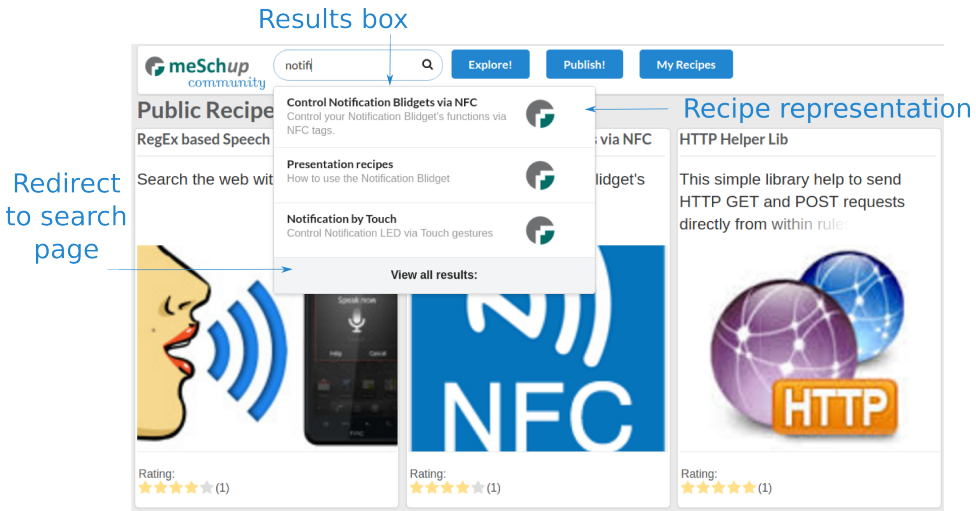

Figure 4.5: Search field with results.

4.4.3 Publishing

The publishing of a recipe is done at a single page. A standard HTML form is displayed, where the user can enter meta information about the recipe as well as choose the

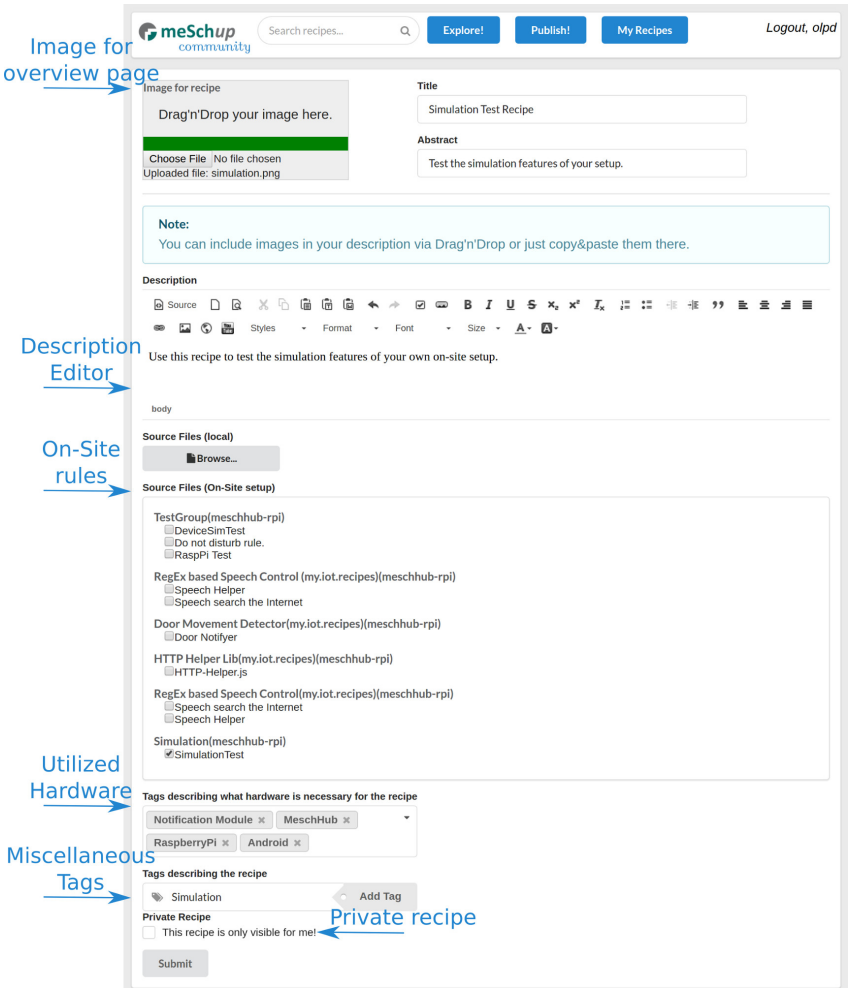

Figure 4.6: Publish form containing all input elements necessary to create a new recipe.

appropriate rules from his on-site setup. The publish form can be seen at fig. 4.6. From top to bottom, the user can specify an image that will appear on the overview page to represent the recipe, as well as a title and an abstract. Beneath that, there is an editor, in which the recipe can be described in further detail. The editor allows to embed images as well as videos from the popular *YouTube* platform. After that, the user can choose which of the rules, currently located at his on-site setup, he wants to attach to the recipe. This function is only available, if the on-site setup of the user is connected to the community server. Below that, further meta information can be attached via a tagging system. The user can specify which hardware he used based on a drop down menu with a fixed set of entries. If he wants to add further information in the form of tags, a field to enter individual tags is located next to the hardware selection. Finally, right above the submit button, the user can choose whether this recipe should only be visible for him (i.e. private) or publicly visible.

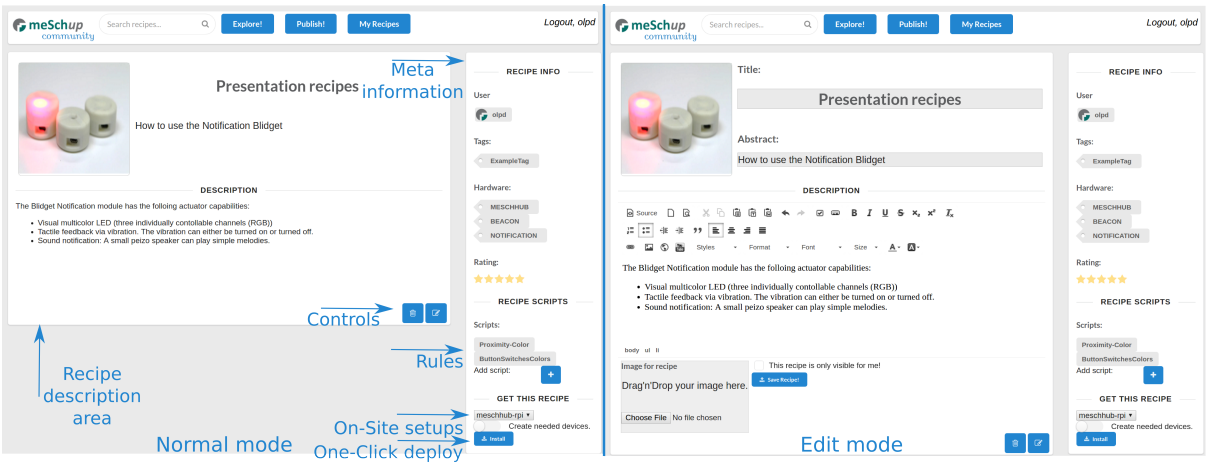

Figure 4.7: On the left side: detailed recipe view, containing meta information. On the right side: same recipe opened in edit mode.

4.4.4 Recipe Detail View

The recipe detail view shows all recipe related information at one place (see fig. 4.7 on the left side). In the center, a large area is determined for displaying the description of a recipe. Embedded images and videos will be displayed in that area. At the bottom, it contains two controls. The first, a *delete* button, represented with a trash bin icon, to delete a recipe. The second, an *edit* button, which enables an edit mode if pressed. Both operations are only available to the user, who created the recipe. When starting the edit mode (fig. 4.7, right side), the description of the recipe is replaced by the same editor used to originally create the description. Besides the description, the recipe image as well as its title and abstract can be modified.

On the right side of the description area, a meta information area is visible. It is split into three regions. The top region shows general information about the recipe: the creator of the recipe, tags associated with it, and the current rating. Clicking a rating star will rate the recipe with the selected number of points.
The region in the middle contains the rules that are attached to this recipe. Clicking on one of the rules relocates the client to a page, on which the content of the rule is displayed. Additionally, new rules can be added via the plus icon.
The bottom region shows the currently connected on-site setups in a dropdown menu. Deploying the recipe on the selected setup is done by the *Install* button. Activating the checkbox between the *Install* button and the setup dropdown, will create all the devices used in the attached rules, on the deployer's on-site setup.

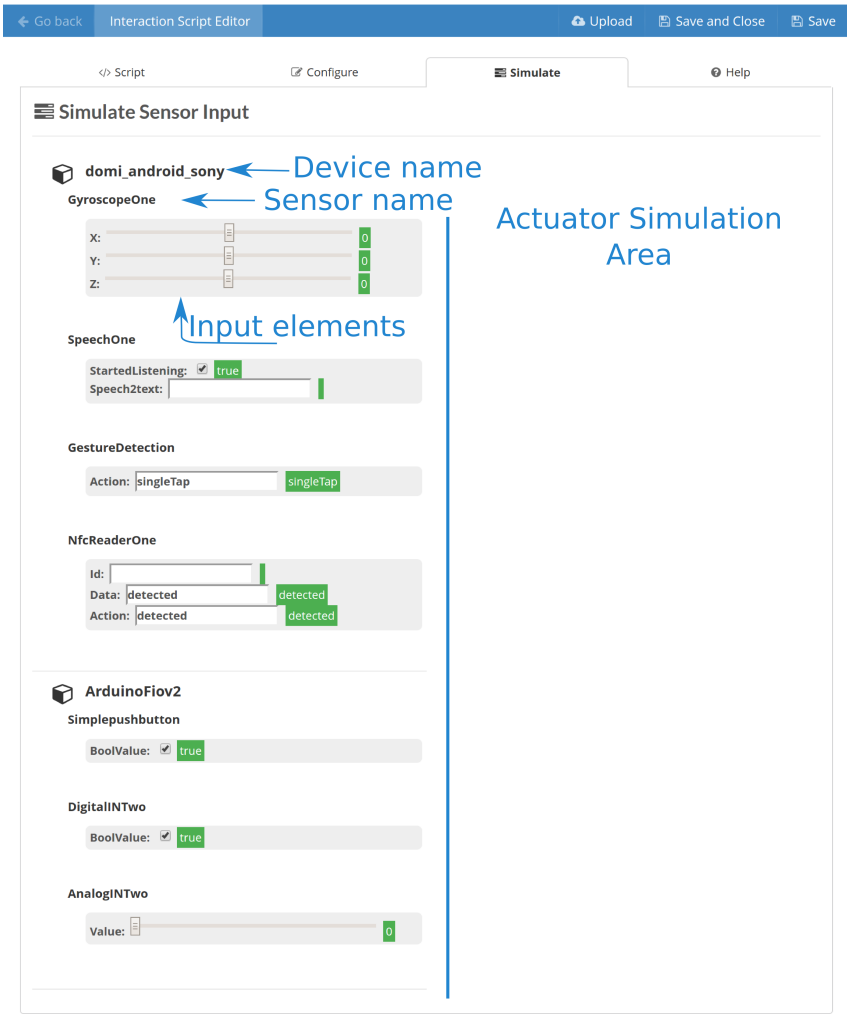

Figure 4.8: Simulation view on an on-site setup.

4.4.5 Simulate

The simulation feature created in this thesis, is implemented at the *meSchup* server (on-site setup). Its functionality is described in section 4.6.6. An example screenshot of the simulation page, can be seen in fig. 4.8.

The top shows a blue navigation bar that allows the user to go back to the previous page or save his current progress of creating a rule. Below that, four tabs can be seen. The first tab redirects the user back to the rule he is currently working on. Tab two was not in use at the time this thesis was written. The fourth tab shows a screen with helpful information when creating rules. Below the tabs is the simulation screen, it is divided in two parts. On the left half of the screen, the devices and corresponding sensors that are

used in a rule are visible. Every sensor has a number of input elements that represents the data, which is generated by this sensor. Moving a slider or pressing the *enter* key in a text input will create a new sensor event, consisting of the current input data.
The right half is designated to show the corresponding actuator events of the system.

4.5 Backend

The backend handles incoming HTTP requests. It returns the corresponding website to a request, and provides an API to retrieve data from the database.

4.5.1 Database

As stated above, *CouchDB* is used as a database. It stores all recipe related information except images. A single entry in the database represents a deployable and shareable recipe. *CouchDB* was chosen because of various reasons.

Firstly, *CouchDB* is fully accessible via a HTTP interface. All database functions can and have to be accessed via the HTTP REST interface, there is no native solution. In the case of the smart home and IoT environment this thesis is based on, this is an advantage. Future projects that extend the current scope of the *meSchup* platform can make use of the community database. There is no restriction of technologies since every device that is capable to communicate via HTTP can access the community database.

Secondly, it is a document-based database model, which is especially suited for the intended purposes. Since the community platform developed during this thesis is merely a prototype, changes to the data model during development were inevitable. Due to the document-based nature of *CouchDB* changing the data model is very easy. Every recipe in the database is a single document. One advantage of document-based databases is that documents within the same database do not need to have the same structure. This means that changing the data model for new recipes does not have any impact on previously stored recipes. Due to that, future changes and extensions of the community platform that append information to the recipe data structure do not impact already existing recipes.

Thirdly, a feature of *CouchDB* databases are so called “attachments”. Attachments are files that can be *attached* to any document in the database. They are useful when the attached files need significant memory and are not required at every document

download. In the case of the community platform interaction scripts are stored as attachments. For the visualization of a recipe, the attached interaction scripts are not necessary. Hence, storing the scripts as attachments and downloading them only when the user actually needs the data, saves bandwidth and improves responsiveness of the user interface. *CouchDB* provides the option to include attachments of a document when requesting it. Therefore, a single request is sufficient to retrieve all recipe related information, including interaction scripts, when needed.

Fourthly, keeping multiple versions of the same document is easily applicable with *CouchDB*. Every document in the database receives an unique identifier to distinguish documents from each other. Furthermore, all documents have an integrated continuous revision counter as well. This counter is used to keep track of updates of documents. Different versions of the same document can be distinguished based on the revision counter. Implementing a history of recipes is therefore simple. Updated recipes only need to be saved as another recipe in the database. Based on the unique identifier as well as the revision counter one can extract old versions as well as the most recent version of a recipe based on their revision counters.
Another advantage of the revision counter in *CouchDB's* documents is its write consistency. A document update is only possible on the latest revision. Therefore, if two users try to update the same recipe at the same time, *CouchDB* will abort one update. This means that concurrent updates will not break the consistency of the database.

Finally, *CouchDB's* search functionality is very powerful. To search for documents in *CouchDB* one has to write a so-called *view function*. This function basically maps a key to a value. In this case, a key can represent a part of an attribute of a document (for example a word from the title), while the value would be the complete document (i.e. the recipe itself). The same key can also be mapped to various values. *CouchDB* does automatically create an index of those key-value pairs to speed up searching.

> **Example:** *To search in the title of a recipe, one would create a view function that maps every single word of the title of a recipe to the actual recipe. If two different recipes have identical words in their title (for example "bluetooth"), then the index would contain the key "bluetooth" twice, once mapped to the first recipe and once mapped to the second.*
> *Issuing a search for the keyword "bluetooth" to the CouchDB database, would then yield both recipes.*

Therefore, searching a *CouchDB* database is very fast since the actual computing overhead (i.e. creating the index) is only done once for every created document. Every subsequent search for a keyword can use the already existing index to lookup the corresponding result.

4.5.2 API

To offer the client a possibility to retrieve data regarding recipes, a RESTful HTTP API is provided that consists of six main resources. The naming convention of resources can be seen in fig. 4.9. Each main resource logically groups functionality to provide

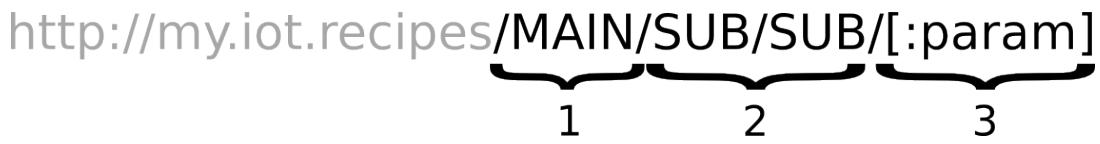

Figure 4.9: The URL naming convention. *1* and *2* represent the main and sub resource respectively. *3* shows a query parameter.

an interface that is easy to use for future developers. Further sub resources represent individual functions. Query parameters are used to add information to a request. Both, sub resources and query parameters are optional and can be used in arbitrary numbers. Depending on the functionality, requests to a certain resource might need to be attached with additional data. For example, when creating a new recipe, the client sends its request to: `http://my.iot.recipes/recipe/add` attached with a JSON object in the body that includes all necessary information.
The following gives an overview of the functionality of each main resource as well as an exemplary outline of the included sub resources. A full description of the API created for the *meSchup* community website, would exceed the scope of this thesis.

`/users`

Combines functionality regarding user accounts. It provides the means to create new users as well as session handling.

`/users/signup` To create a new user, requests target this resource with additional information, such as a users email address and password, in the body.

`/users/login` Login requests target this resource including the username as well as the password of a user.

`/users/activate/[:key]` To prevent the community from being flooded with fake accounts, a new user needs to activate himself by accessing the given resource. The additional

parameter is a key, randomly created by the server when adding the new user to the database.

`/recipes`

Provides functionality regarding a list of recipes. Retrieving all or multiple recipes is done by using this resource.

`/recipes/` Will result in a response that includes all publicly visible recipes in a JSON object.

`/recipes/search/[:input]` Provides all recipes that match the specified search query.

`/recipes/user` Provides all recipes that the currently logged in user created.

`/recipe`

Congregates all functionality that has effect on a single recipe.

`/recipe/attachment/new` When an additional attachment needs to be added to a existing recipe. The request has to target this resource including the content of the attachment.

`/recipe/add` Creating a new recipe is done by sending all necessary information to this resource.

`/recipe/rating` Provides functionality to rate a recipe. The name of the rating user as well as the rating score need to be in the request's body.

`/onsite`

Contains all functionality that targets the on-site setup of the user, or retrieves information thereof.

`/onsite/add` When a recipe needs to be deployed at a user's on-site setup, a request to this resource is necessary. The request has to contain information about which recipe needs to be deployed as well as the name of the target setup.

`/onsite/recipes` Retrieves all rules and rule groups on a user's on-site setup.

`/onsite/connectedServers` Retrieves a list of all on-site setup of a particular user, that are currently connected.

`/upload`

Handles file uploads, such as images or script files. Uploaded images are stored at a publicly available folder on the web server. Script files are added to the corresponding recipe as an attachment and deleted afterwards.

`/upload` This resource is targeted when files need to be uploaded to the web server.

`/codeview`

Used to retrieve the content of an attachment of a particular recipe.

`/codeview/[:recipeId]/[:attachmentName]` Retrieves the content of the attachment specified with the `attachmentName`, which is part of the recipe with the identifier `recipeId`.

4.5.3 Cloud Deployment

The developed community platform is hosted at *Amazon Web Services* using *Elastic Beanstalk*. This is a *Platform-as-a-Service* management system, which allows for easy deployment of web applications. A cloud deployment was chosen to be easily scalable in the future as well as to avoid maintenance efforts during development.

4.6 Functionality

The important features of the community website, created in this thesis, are stated here.

4.6.1 Connectivity

Different transmission mechanisms are used to realize the features of the community website (see fig. 4.10). For users to access the available recipes, HTTP is sufficient. However, to retrieve data, such as rules or device data, from an on-site setup, using HTTP is not possible.

When a user deploys a recipe on his on-site setup, the community web server needs to access the setup in order to transmit rule and device data. There are two main issues, that prevent this transmission: Firstly, HTTP is a request-response method of communication, which means that for the server to transmit data, a request from the client has to precede. The server cannot send data to the on-site setup at an arbitrary time without a request being send previously.
Secondly, since most *meSchup* setups will be installed at a user's home or office, this introduces another issue. If the on-site setup is installed behind a router in the user's home, it might not always be accessible from the outside. Most modern home routers are equipped with firewalls and NAT (network address translation), which might prevent connection attempts from the outside.

To avoid this issue, a *WebSocket* connection is used. The connection is first established from the user's on-site setup to the community server, to prevent firewall intervention. Furthermore, the connection is not terminated after a successful transmission of data, therefore the server can continue sending data to the on-site setup.
The *WebSocket* of the on-site setup is, in general, a wrapper interface for the local HTTP API. Incoming messages at the on-site setup have to be of a form, which allows sending them directly to the local HTTP API. The corresponding response is then sent back via the same *WebSocket*. Due to this, there is only one API that needs to be maintained. Furthermore, basically all of the operations that are accessible via the HTTP API from the local network of an on-site setup, are also accessible via the *WebSocket* connection.

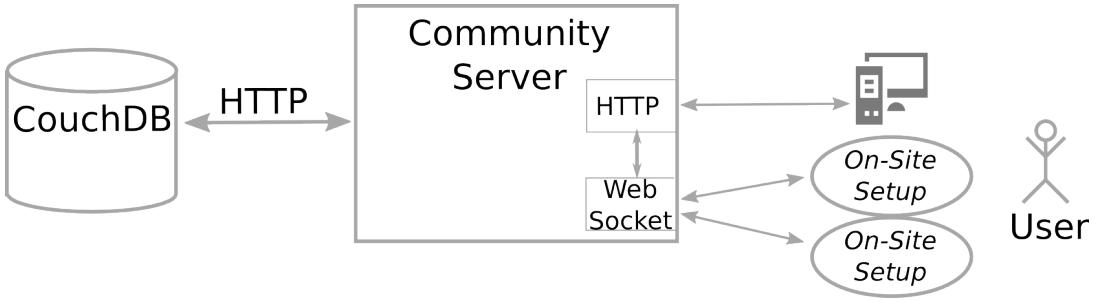

Figure 4.10: Connections utilized for the *meSchup* community.

4.6.2 Search

Searching available recipes is a vital feature for users to find recipes. As described earlier (section 3.5.4), users should be able to search in various attributes of a recipe. Therefore, a text based search was implemented, that includes the title, abstract, description, tags and hardware information of a recipe.

It is implemented using the powerful key-value search functionality of *CouchDB* (see section 4.5.1). *View functions* have been implemented, that create key-value pairs for the above mentioned title, abstract, description, tags and hardware information. To enable searching for partial words, the *view functions* even maps every substring of an attribute to the respective recipe. This means, searching for the word “blu” will also return all recipes that, for example, contain the word “bluetooth”.

4.6.3 Recipe Publishing

The data exchange in order to publish a recipe is shown in fig. 4.11. At first, a user has to visit the corresponding web page, on which he can create a new recipe. Upon visiting this website, the community server requests data from the user’s on-site setup about the currently installed rules. This data is used to build the website giving the user the choice which rules he intends to publish. After adding descriptive meta information and selecting the appropriate rules, the user submits the created recipe to the server. After the server handles the *device retrieval* (see next section), the recipe is saved and a response is back to the client. The response includes information about whether the operation was successful. In case of a success, the client is relocated to a new website, showing the newly created recipe.

4.6.4 Device Retrieval

When a user creates rules at his on-site setup, these rules, in general, utilize the devices that are available for a particular setup. If these rules are published at the community website and then downloaded by different users, the utilized devices might not be available or have a different name. The issues that arise due to this, as well as a possible solution, is discussed in section 3.5.9.
The solution approach was about mapping the devices that are needed to those available at the deploying user's on-site setup. In order to do this, an important step has to be done while publishing a rule. This step is the *device retrieval*, which gathers data from the creator's on-site setup about the devices that are used in a particular rule.

> **Example:** *User C creates a rule that utilizes an Android smartphone and various sensors as well as actuators of it. Additionally, the rule uses a smart wall plug in order to control a TV that is connected to it. After finishing the rule, user C creates a recipe containing the rule at the community website.*
> *User D finds the recipe and deploys it on his own on-site setup. D possesses an Android smartphone and a smart wall plug as well.*

When user *C* creates the recipe, the community server attempts to retrieve data about the smartphone and the wall plug utilized in *C's* recipe. This data is important, because it contains the configuration of the sensors and actuators of the devices. A sensor might be configured to send more or less frequently data, whereas an actuator, such as speech recognition, might need to be configured for a certain language. When deploying a recipe, this information is important in order to configure the devices of the deploying users correctly.

The data exchange, which is necessary to create a recipe, including device retrieval, is also depicted in fig. 4.11. Implementation wise, when a user visits the web page, on which he can create a recipe, at first the corresponding rules are retrieved from his on-site setup. Afterwards, regular expressions are used to search for device names as well as the names of sensors and actuators, in the source code of the rules. When finished, for every device name found, the on-site setup receives a request to transmit the corresponding data. The data is then filtered in a way, that only the configurations of the actually utilized sensors and actuators remain. After that, the device data is attached to the recipe, that is currently created, and saved with it in the database.

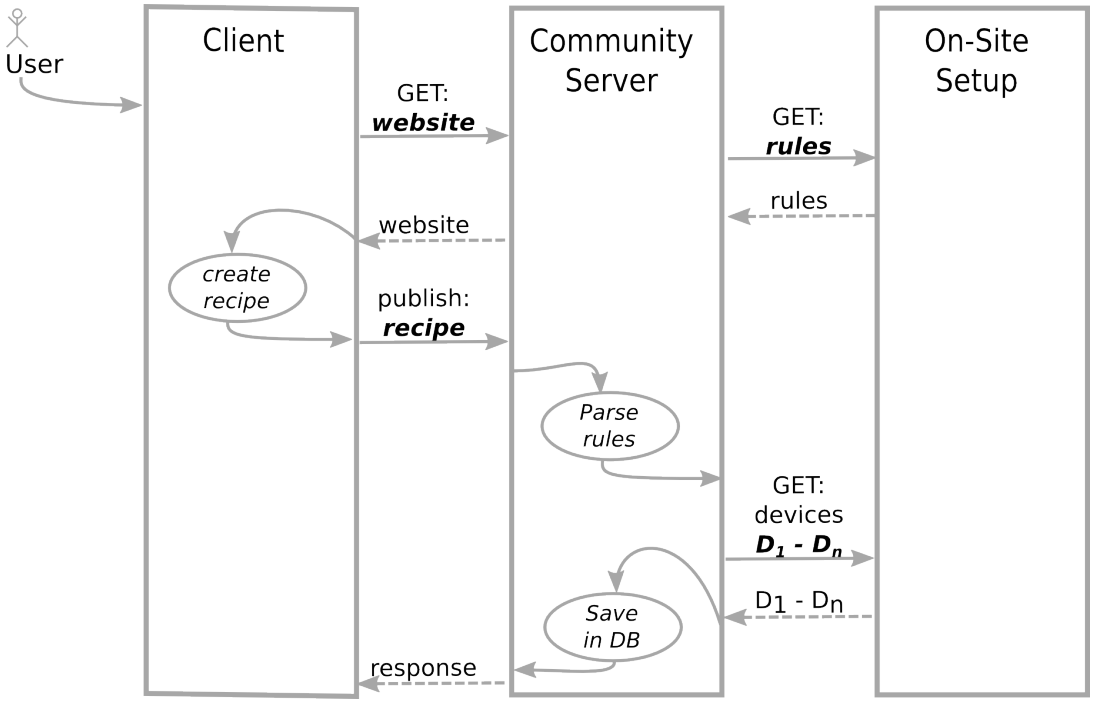

Figure 4.11: Communication during the creation and publishing of a recipe. Solid arrows mark a request of data, dotted arrows represent the response. Due to readability, login messages are not shown.

4.6.5 Deployment

Besides the *device retrieval*, the *deployment* of a recipe onto a user's on-site setup is equally important. The sequence of actions and the data exchange is shown in fig. 4.12. Before deploying, a user has to visit the community website. He chooses a recipe to be deployed as well as one of his smart environment as the target (for details, see section 4.4). Finally he initiates the deployment of the recipe.
When the community server receives the request to deploy a recipe, various data needs to be sent to the user's on-site setup. Firstly, a new rule group, containing the name of the community platform, is created at the on-site setup. Secondly, for every rule that is attached to the recipe, a request is send to the on-site setup. Each request contains a rule as well as the target group where it has to be created in (the new created rule group). Thirdly, for every device that is saved within the recipe, another request is send to the on-site setup. These requests contain the device data, which was retrieved during the publishing (see section 4.6.4). The *meSchup* platform creates new (virtual) devices, based on these messages. The creation of devices is optional and can be de- activated at the community website for every deployment (see section 4.4).

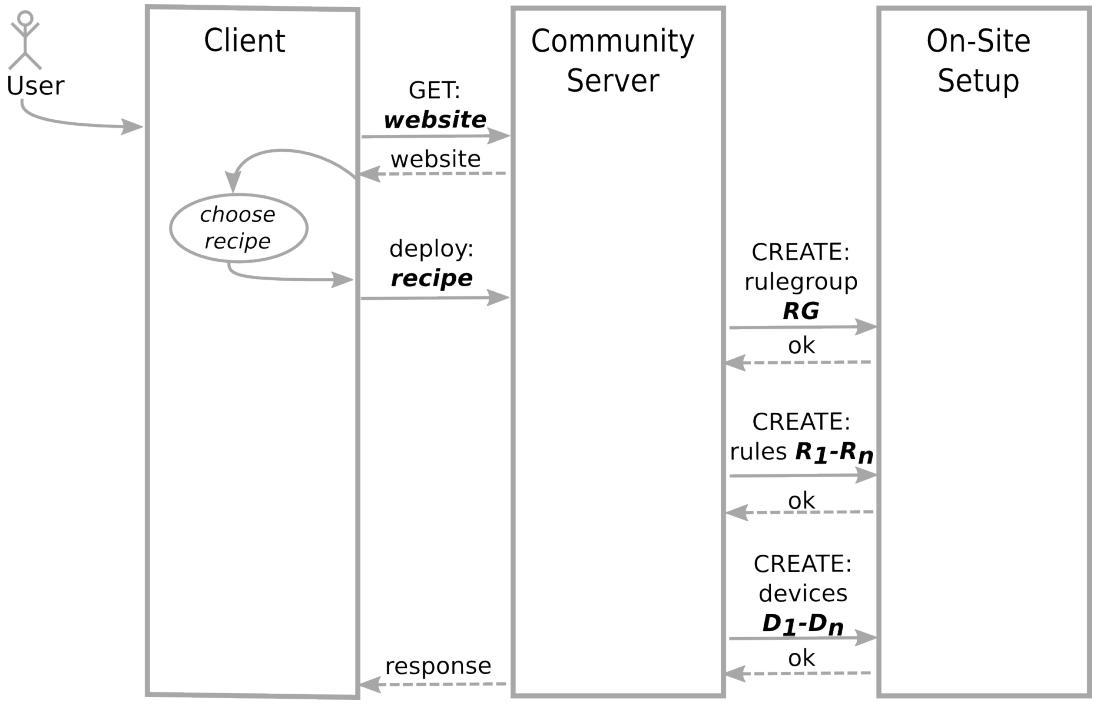

Figure 4.12: Communication during the deployment of a recipe. Due to readability, login messages are not shown.

4.6.6 Simulation

Another important feature implemented during this thesis, is the simulation of sensor data. It is a feature of the on-site setup of a user and allows to artificially inject sensor data into the system. While various use-cases exist for this, the two main applications (debugging and testing recipes without full hardware support) are described in the following.
On the one hand, it allows to create sensor data that is physically complex to produce. For example, if a sensor of the user's environment reads atmospheric pressure, it is challenging to manually change its readings. The simulation feature hereby helps generating sensor data manually in order test rules while creating them.
On the other hand, when a user downloads recipes from the community website, he might not have all sensors in this smart environment, which the original creator used in this recipe. So, for the user to still be able to see the results of the recipe in action, he can simulate the missing devices by injecting their respective sensor readings manually. This way, a user is able to test recipes in order to further decide whether to get additional hardware.
This feature is a part of the simulation idea described in section 3.5.14.

The implementation of this feature is realized using a two step process. First, the respective rule is parsed via a regular expression, to find all utilized devices as well as the sensors in use. After that, the user interface, in which the user can manually inject events, is dynamically created based on the results of the regular expression. For every sensor that is used in the respective rule, detailed information about it is retrieved from the API offered by the *meSchup* platform. Using the retrieved data, HTML input elements are generated.
For example: The detailed data about an accelerometer-sensor of a smartphone indicates that its sensor value consists of three floating point numbers each in a range of $[-1.0, 1.0]$. Further information indicates, that these translate to the axes of a coordinate system, namely X, Y and Z. Based on this information, a slider for each axis is generated with the respective limits. Furthermore, the implementation ensures that each movement of a slider will trigger the sending of new artificially created sensor data.
The same procedure is done for all sensors and their respective data types. Sensors that produce text data are rendered as string inputs, boolean sensors are rendered as checkboxes and integer data is rendered as a slider as well.
Once the HTML of all devices and their sensor has been created, it is attached to a website of the on-site setup, so that the user can interact with it. The visual result of this feature can be seen at fig. 4.8.

4.7 Recipe

For a user of the community platform, a recipe contains all information needed to deploy certain behaviour onto his on-site setup. Even on the implementation side of the community platform, a recipe is a centralized data structure that holds all information necessary for deployment and visualization on the website. To the date of this thesis, this is a novel approach for distributing behaviour between smart environments.
The recipe itself is a JSON object that can easily be attached with additional data. For an example, see fig. 4.13

4.7.1 Advantages

There are multiple advantages in storing recipe related information in a JSON object at one place instead of distributing it across files or databases.
Firstly, having all data combined simplifies the retrieval. This way the community website can retrieve all information with a single request. Also, future systems that will use the *meSchup* environment and possibly the community platform can benefit from the simple retrieval.

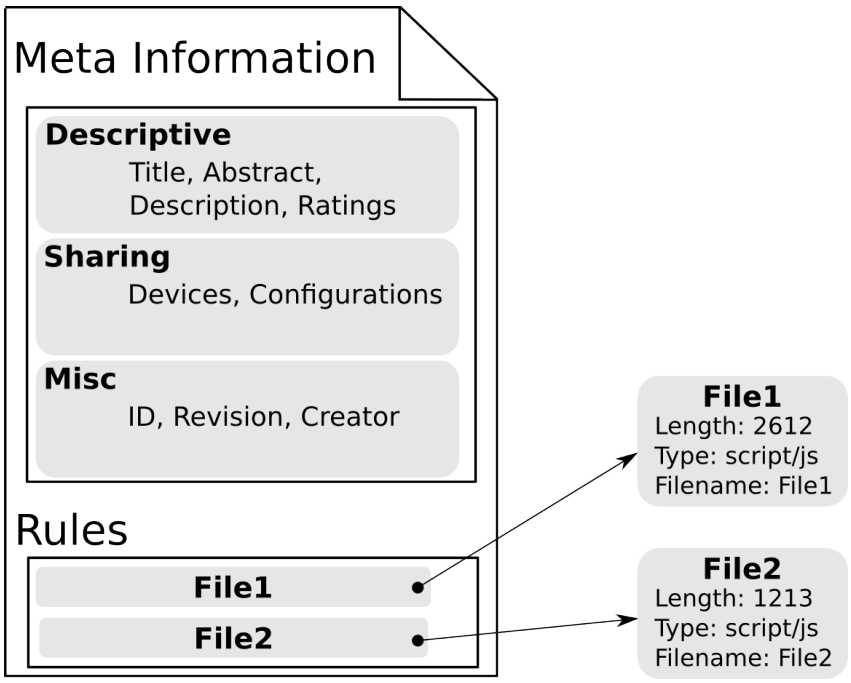

Figure 4.13: Sample representation of a recipe.

Secondly, due to the JSON structure of the data, recipes can be served as files. A recipe can be represented as a single JSON file and therefore easily be handled by third party software.
Thirdly, versioning as discussed in section 4.7.3 can be realized easily. Since all data is centralized, complex queries to create a new version of a recipe are not necessary. Furthermore, recipes could simply be stored as regular JSON files and therefore be backed up very easily.

4.7.2 Content

The recipe holds different kinds of data: meta information for the description of a recipe online, interaction scripts and information about the devices utilized in it. In order to distinguish different recipes, a unique identifier is part of the data model. Furthermore, every recipe has a continuous revision counter to distinguish different versions of the same recipe.

Devices

When creating a new recipe, the specified interaction scripts are parsed to identify the devices utilized in the scripts. With this information, the community platform tries to retrieve specific information about the devices as well as their current configuration from the user's on-site setup. In case another user wants to deploy the recipe, this information is important to create and configure the necessary devices on his on-site setup.
The data structure retrieved from the user's on-site setup is a JSON object as well. To achieve full compatibility with the functions of the *meSchup* server (the user's on-site setup), the retrieved data structure is added to the recipe unchanged. It includes descriptive information about a device, such as the name and manufacturer as well as links for further reading. Moreover, it contains detailed information about the sensors and actuators of each device. Especially about the data structures that each sensor generates and that each actuator requires to change its state. This means, with the information retrieved about a device, one is able to deduct how a sensor and actuator event is composed before retrieving it. This is important for the simulation feature described in section 4.6.6.

Interaction Scripts

Another important part of a recipe are the interaction scripts a user loads up. Those are stored as *attachments* in a recipe. Attachments are a special data structure used in *CouchDB*. For details about the database used in this thesis, please see section 4.2.1. Attachments have the advantage, that they can easily be omitted when accessing a recipe in the database. This reduces the amount of data that needs to be transferred to a user's browser in order to view a recipe. Attachments can be transmitted within a recipe when needed, but can also be transmitted individually, for example when rendering the code view of a recipe.

Meta Information

Besides the technical necessary information to share the behaviour that is intended with a recipe, it also includes meta or descriptive information. Such information is used to visualize recipe on the community website. It includes the title, abstract, description and which user created it.
Furthermore, it includes information about the rating and whether the recipe is private (i.e. only visible for the creator). The rating is a list of entries, in which each entry specifies which user rated the recipe and how many points he awarded.
Due to *CouchDB* being document-based, extensions such as commenting features are

easy to implement with the current data model. In case *commenting* becomes a necessary feature for the community platform, comments regarding a recipe can easily be attached to the existing recipe object.

4.7.3 Versioning

Due to the centralized data model of recipes versioning is simple, compared to a distributed data model. When an update of a recipe is necessary, an internal revision counter is increased and the updated recipe is saved with the the new revision number. Using this revision counter, multiple copies of the same recipe (with different revisions and data) can be stored. So in order to provide the history of a recipe, the versions of the recipe with lower revision number need to be retrieved.

5 Evaluation

In the following, details about the conducted study will be discussed.

5.1 Evaluation Methodology

As an evaluation methodology, *in-the-wild* deployment was used. It has various advantages compared to a regular user study, in which the participants only use the system for a short amount of time in order to fulfil a task.
Mainly, the participants have the opportunity to use the system for a longer time, thus they get used to it and can experiment with every feature. Additionally, when participants work with the system at home, information about a real world application of the system is gained. Opposed to a “clinical” setup in a laboratory where the environmental influences do not change.

5.1.1 Workshop

At the beginning of the study, a workshop was held where participants were introduced to the *meSchup* and community platform.

Goals

The workshop aimed to achieve several goals. Firstly, demonstrating how the *meSchup* platform works, so that the participants were able to understand and create new rules for it. Secondly, showing the features and usage of the community platform so that a sharing of recipes during the study is possible. Thirdly, demonstrating and explaining the utilized hardware, so that the participants familiarize themselves with the capabilities of the platform and the hardware. Fourthly, utilizing the first impressions of the participants to gain feedback about both, the *meSchup* platform as well as the community website. Furthermore, participants feedback should be used to find out what features about both

platforms they would like to have in the future.

Activities and Content

The first part of the workshop was an introduction into the *meSchup* platform. Participants were explained what the platform was designed for and who developed it.
After that, participants were told how the platform works and about which aspects they have to care about when creating rules. That included an introduction into the event-based messaging system of *meSchup* as well as further information about how the created rules are handled by the system.
Next, they were shown a live demonstration about how to create rules on one of the participants setups.
Afterwards, the community website was introduced. The participants learned about the goal of the community website and how it is integrated in the *meSchup* platform. Furthermore, they were shown how to use it in order to publish their creations and how to deploy recipes of the other participants.
To complete the introduction into the platform, the next topic of the workshop was the setup the participants would receive for their home use. They were explained which devices they would receive, their capabilities and how they could interact with these devices.
After the introductory phase, the participants gained some practical experience with the platform. In order to get familiarized with the platform, they were told to add their own Android smartphone to their respective setup. Then, they received the task to implement a rule, using the hardware of the study, on their own *meSchup* setup. The task was to switch a WiFi-controllable wall plug on and off, using the hardware provided for the study.
Once the participants finished the task, a group discussion took place in order to gather feedback on the platform and possibly new features. For the results of that discussion, please see section 5.3.1. Finally, to increase the participants motivation in building their own smart environments, they received additional sensors in order to be able to build the devices they mentioned during the group discussion.

5.1.2 Setup

The study runs over the duration of two weeks. Three persons participate in the study, each of them receives one set of hardware to use the *meSchup* platform as well as the community website. All of the participants have a computer science background. Two of them are studying software engineering at the time of the study, the third is a

self-employed developer.

The hardware that every participant can utilize during the study, is described in the following as well as an image of the hardware can be seen at fig. 5.1.

MeschHub The MeschHub is the central controlling point of the smart environment. All generated sensor data is sent to this instance and actuator data is sent from it. The *meSchup* server software runs on it (see section 4.1.1). All *MeschHubs* were configured identically before the study started. A variety of rules and examples were installed to provide the participants with useful code and further simplify the familiarization with the platform.

Raspberry Pi A Raspberry Pi with specialized software installed on it. It can be connected to a display or beamer, while the content it shows can be configured via rules. Furthermore, it can act as a Bluetooth Low Energy scanner and therefore detect devices in another room than the *MeschHub*.

WiFi Smart Plug The smart plug is a WiFi controllable wall plug that can be used to switch connected devices on and off.

Notification Blidget The notification blidget is a small, battery powered micro controller with the purpose of notifying users. It contains a freely controllable multicolour LED, a vibration motor and a piezo speaker to play sounds. Moreover, its intelligent energy supply allows the *notification blidget* to be switched on when it gets moved. After being switched on, it stays on for a configurable amount of time before it changes into an energy preserving sleep mode. While it is switched on, it uses Bluetooth Low Energy to communicate with the *MeschHub*, and sends heartbeats to show its availability.

Button Blidget The same technology as the notification blidget, but instead of notifying a user this blidget contains a button. When pressed and released, the blidgets send the appropriate information as sensor data. Due to a shortage of button blidgets, only two participants received those.

Blidgets (2) Blidgets are coin-sized micro controllers, that use Bluetooth Low Energy to send announcements at a regular interval. Using a corresponding Bluetooth scanner, a user can determine whether a *Blidget* is within range of the scanner, and estimate a rough distance.

NFC Tags (2) NFC-Tags, in the form of stickers, provide an unique identifier when being scanned with the appropriate device. The Android smartphones of the participants have NFC-scanners and the retrieved information can be used further in creating rules.

Android Smartphone All participants use their own, or a supplied smartphone. Its variety of sensors and actuators can be used by the *meSchup* platform.

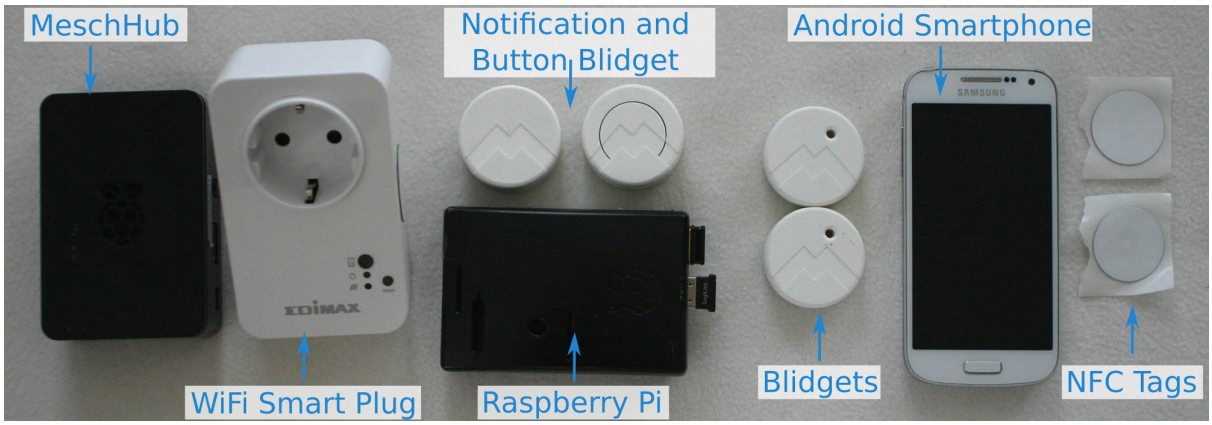

Figure 5.1: An overview of the hardware, which was utilized in the study.

5.1.3 Tasks

Since the intended community platform enables users to share recipes with each other, the usage of that functionality should be investigated in particular. Therefore, before the study starts, three tasks are designed and at the first announcement, each participant is assigned a different task. The announcement of the tasks has to be in a private manner, for example via email, so that the participant do not know that they work on different tasks. At the next announcement, the tasks change, so that every participant gets a new task, but again, all participants have a different one. Due to this, all participants have different tasks amongst themselves throughout the study, and every participant has to work on every task.
During the initial workshop, the participants are told to freely use every recipe that is available on the community platform, that could help them solving their tasks. Furthermore, they were encouraged to share their created rules - both for tasks and independently developed ones - on the online community website. In case the participants utilize the platform to exchange their work, at the end of the first assignment, solutions to every task could be found online. Therefore, all following tasks could be solved by the participants by downloading the corresponding recipe and modifying the deployed rules to fit their needs.

The tasks that are assigned during the course of the study, are explained in the following.

Task 1

The participants are challenged to use the speech recognition functionality of their smartphones. Based on the example rules installed on their *MeschHub*, the participants are asked to create rules, that allows interaction with their hardware via speech recognition.

The first subtask was to control all of the *notification blidget's* functions with speech input.
This way, the participants learn about how to use existing rules in their own creations.

The second subtask was an advanced version of the previous one. Participants should use the speech input to control a display, connected to their *Raspberry Pi*. The connected display was supposed to show websites that were recognized by the speech input.
The intention of this task was to make the participants understand, how two completely separated devices can be used to enrich their smart environment with useful functionality.

Task 2

With the lowering of standby energy consumption in mind, the WiFi controllable wall plug should be used to turn off devices that are not used. This task was supposed to show the characteristics of the *meSchup* platform being an event-based system. Every time a new sensor event is received by the *MeschHub*, all rules are executed. One key characteristic of the platform is, that based on the event-drivenness, local variables created in a rule are not stored between two individual rule executions. To solve this problem, the platform offers a special data structure, that is available across rule boundaries and can store data as long as the server is running. The participants are supposed use this functionality to store data.

The first subtask was to use a *Blidget* as well as the *Raspberry Pi* to control the wall plug based on the proximity of those two devices. The wall plug should be switched on, as long as the *Blidget* is within close range of the *Raspberry Pi*, otherwise it needs to be switched off. To be more resilient against transmission interferences that are common when using Bluetooth Low Energy, the participants were asked to use the platform's functionality to save data across rule executions. Therefore, past data of the *Blidget's* signal strength had to be used to control the wall plug. This way, short interferences have no effect and do not switch the wall plug mistakenly.

This task ought to show the users the functionality and capabilities of the cross execution storage of the platform.

The second subtask aimed to solve the same goal (switching the wall plug), with a different type of interaction. Instead of proximity information based on a *Blidget*, speech recognition from a smartphone had to be used.
This task should offer the participants ways, how to use different types of interaction to control the behaviour of their smart environment, as well as familiarize them with another type of interaction.

Task 3

The participants are challenged to use the intelligent energy supply of their *notification blidget* to detect movement. Since those blidgets start sending heartbeats once they are moved until they fall in a sleep mode, movement can be detected by awaiting such a heartbeat.

The first subtask was attaching the *notification blidget* to an object, of which movements the user would like to be informed. The participants were informed that the wireless range of the blidget is limited, and that the receiving hardware should be within a reasonable range. Once movement was detected, the participant was asked to choose an appropriate form of notification, and show the movement. Possible usage examples were a letterbox and entrance doors.
The participants should learn about the usage scenarios of *notification blidgets* as well as different notification types.

The second subtask reversed the usage of the first task. Instead of waiting for an object to move, the participants should attach the *notification blidget* on a typically moving object and create a notification if there is no movement for a specified time. An example usage was the attachment of the *blidget* on their desk chair.
As in the first subtask, the participants should learn about the usage scenarios of the smart *blidget*. Furthermore, they had to use the platform's functionality of storing data across rule executions, for example to find out when the last movement of the *blidget* appeared.

5.1.4 Data collection

During the course of the study, several sources of data are used. Firstly, there is a web blog set up for the study, where the participants can exchange ideas and comments with

each other. They were asked to use the blog to announce their ideas and problems they encounter for future consideration.
Secondly, on the community server a logging mechanism was implemented. This mechanism records, amongst others, how many recipes are created and deleted, as well as how many rules are uploaded to the community website. This mechanism is used to get quantitative feedback about the study.
Thirdly, two group discussions, at the start and end of the study, are used to gather ideas and suggestions from the participants.
Finally, an individual interview with every participant after the first half the study should bring additional insights in the users wishes and suggestions regarding the platform.

5.1.5 Timetable

The study itself lasts two weeks. Participants are asked to solve three tasks during that time, which will be announced during the course of the study. Additionally, the feedback of the participants is gathered at three times during the study. Below is a list of announcements and meetings that happen throughout the study.

1st day	Workshop, Announcement of first task, Group Discussion.
5th day	Changing tasks between participants.
7th day	Individual interviews with every participant.
9th day	Changing tasks between participants.
14th day	End of study, final group discussion.

5.2 Research Questions

The study is intended to gather mainly qualitative results from the participants. The research questions that need to be answered are described in the following.

5.2.1 *Q1: Are users (creators) willing to share their creations?*

An important question to answer, is whether users are actually willing to share the rules they created for their smart environment. Especially if they are not willing to share those it is important to know why, in order to find ways to motivate them to share.

5.2.2 *Q2: How do security and privacy aspects influence sharing and using of recipes?*

Another equally important question is how security and privacy influences sharing and using of recipes. With a better understanding about why users might be hesitant to publish or deploy recipes, better security mechanisms can be created to protect users of the platform.

5.3 Results

During the course of the study, the participants feedback was sought on multiple occasions. The results of the discussions and interviews, as well as a summary of the quantitative data gathered during the study, is described in the following.

5.3.1 Group Discussion during Workshop

The group discussion showed, that the participants are willing to share their creations on the community platform. Furthermore, they thought that the recipe sharing is a very important approach of distributing behaviour across different smart environments. Nevertheless, during the discussion several privacy related issues were revealed. The summary of the discussion including concerns of the participants as well as feature ideas are presented in the following.

Concerns

The participants mainly stated concerns regarding two security issues. On the one hand, an important discussion topic was the disclosure of sensitive information by publishing rules. For example, if users connect the alarm system of their houses to the *meSchup* platform, the participants stated that the rules controlling the alarm system, might contain sensible information. Therefore, it should not be possible to publish rules that might disclose such information or the possible consequences should be pointed out to the corresponding user.
The same concern was stated when using an electronic door lock that could be controlled via the *meSchup* platform or when rules contain private passwords (e.g. to check an email account).

On the other hand, deploying a malicious recipe and being spied on by somebody was another important topic of the discussion. In case a user deploys a recipe that lets him control the lighting in his home, the information when the lights are switched on and off could be used to find out about the users time of attendance. A malicious developer would create a recipe that lets users control certain aspects of their environment. Additionally, using the scripting language he would retrieve data about the user's behaviour.

Feature ideas

Besides the problem of publishing and deploying recipes of an online community, the participants quickly came up with solutions, that would help prevent such behaviour and encourage members to download recipes.

Blacklisting keywords in rules To prevent users from publishing rules that contain passwords specialized keys to electronic locking mechanisms, rules could be parsed for certain keywords. For example, a user should be notified if his recipe contains the word "password" when trying to publish it. This approach can easily be extended with additional keywords or complex regular expression that can be filtered.

Generalize sensitive information The amount of user specific information that is published should be minimal. When publishing a recipe, individual device names or MAC addresses that are referenced within rules, should be replaced by generic place holders.

Store sensitive data externally One way to prevent the need for clear text passwords or MAC addresses in rules altogether, would be to store such data in a specialized type of rule. This rule would contain user specific data, fox example as a key-value store, and the file or its content would not be publishable.

Policy structure for rights management Introducing a policy infrastructure, similar to the one used in the Android environment, would help to find out what kind of interaction certain recipes need. To be able to access a certain sensor data or send message over the internet, a recipe would need the user to acknowledge specific policies for it. This way, a user knows at all times which capabilities a recipe needs and therefore

can judge whether or not this recipe should have these rights. This idea is very similar to the policy management idea mentioned in the system concept (see section 3.5.13).

Public reputation available Attaching recipes in the community website with publicity data would help users judging whether or not to trust a certain recipe. The participants stated, they would feel more comfortable deploying a recipe when they knew, how often it is in use at the moment as well as an average rating of it.

Simulate actuators An idea, similar to the simulation feature described earlier (see section 3.5.14), was described as well. Judging whether a recipe is trustworthy, could be done by simulating its input and output actions. While the user generates the appropriate sensor data, the *meSchup* platform would simulate the actions of the recipe. Therefore, it would use graphical and textual representations of the recipe's actions. Seeing those, a user could then decide whether or not to deploy the recipe into his smart environment.

Devices users would build

After the introduction in the *meSchup* platform and the group discussion, the users were asked what kind of devices they would like to build, or see as part of the platform. A variety of devices was mentioned, among other things:

Intelligent wall plugs An advanced version of the already available wall plug. It should not only switch connected devices on and off, but measure the power consumption of the connected devices.

Automatic shutters A device that could be integrated into the platform to control shutters on windows.

Augmented reality An improvement of the currently available augmented reality stickers. When captured with a smartphone camera, the sticker's image should be replaced by the image of a website.

Water-level indicator A device that can be permanently installed outdoors and transmits the water-level of a container.

Radio stream player This is an extension of the capabilities of the current *meSchup* Android-App. A corresponding smartphone should be able to play an audio stream when placed on a NFC-tag.

5.3.2 Intermediate Interview

At the start of the second half of the study, each participant was interviewed individually. The questions aimed for feedback on the usage of the community platform, i.e. to gather positive and negative things that the participants encountered. Furthermore, the participants should state what kind of features they are missing or would like to see in the future. Moreover, the participants were asked about the simulation feature that was developed for the *meSchup* server.

Usage

One participant noted that he had trouble setting up his on-site setup and due to a unstable network connection, could not use the publish and deploy functionality of the community website.
Two of the participants stated that they had no issues while publishing recipes and that they were pleased with the simpleness of publishing new recipes. Especially the creation of recipe descriptions and its functionality, which allows adding images via the clipboard, was well received.
At the time of the interview, the participants made use of the publishing functionality, yet none had deployed any recipes created by the others.
It was stated by one of the participants, that he was unsure about kind of information, which should be written in a recipe's description. A style guide describing the contents of a recipe's meta information, or a detailed example should be available for newcomers.
Furthermore, one participant stated that the session handling was erroneous. Instead of showing a warning when a session is no longer valid, the user interface should simply show a login page instead of the warning.
At the time of the interview, none of the participants made use of the simulation function.

Additional features

Besides their findings when using the community website, the users were also asked what features they would like to see in the future.

Publishing device configuration with rules Two of the participants stated, that it would be useful if device configurations would be published with the rules. In case the creator of a rule changes the default configuration of the devices in his environment, users of his work would need these configurations as well. This is very similar to the ideas described in section 3.5.9.

Versioning and updates Two of the participants also said, that the ability to update existing recipes would be useful. The creator of a recipe should be able to apply updates of the attached rules, and that those updates should be accessible by the community. Furthermore, when a recipe is updated, all users that deployed it, should be notified of a new version. Nevertheless, applying updates automatically to a user's on-site setup should not be possible, except the user explicitly acknowledges it. This is similar to the features described in section 3.5.12.

Searching for users An improvement of the search functionality was proposed by one participant. The amount of available recipes should also be searchable by a username. In case a user found a recipe that suits his needs, he might wants to see other recipes of the same creator.

Rating of users In order to judge the publicity and trustworthiness of recipes, one user suggested the rating of other users. In that case, every user would be able to submit a rating of every other user after deploying one of his recipes.

Comments To channel discussions and feedback of a recipe, a commenting function was suggested by one user. This way, users had an simple way of exchanging ideas and talking about a certain recipe. It is an similar idea as described in section 3.5.2.

Signing of recipes To improve the privacy and security aspects of the platform, one user suggested a signing function for recipes. Similar to emails, the creator of a recipe should sign the code and descriptive information of his recipe. Based on the digital signature, users who know the creator in person or via previous communication can then ensure that the creator is the person he represents. For users not knowing the creator, at least the authenticity of the recipe can be ensured.

5.3.3 Group Discussion after Study

At the end of the study was a group discussion with two of the participants. Additionally, there was an interview with the third participant. For the sake of simplicity, the results of both events will be described as one and referenced as the *final discussion*.
The final discussion yielded diverse feedback. The participants introduced ideas for new features, objected to some difficulties during the study and repeated their request for the features they established during the first two discussions.

Usage

At the time of the final discussion the participants worked on all tasks and had come in contact with the publish and deploy functionality of the community website. One of the

participants still struggled with an unstable network connection of his on-site setup and therefore was not able to publish or deploy any recipe. The other two participants noted that they had successfully worked with the community website. Besides one participant, none made use of the simulation functionality in order to develop rules.

The participants were overall pleased with the usage of the *meSchup* platform in general and the community website. Only two issues were stated during the discussion. Firstly, the API, which is used to create rules, is not fully consistent. To access functionality on different devices, a diverging naming convention is used. The participants stated that an global naming convention and a consistent approach of accessing information should be available.
Concerning the community website, the participants noted that the visibility of recipes should be improved. While the overview page offers a well-arranged set of recipes, a single recipe representation should be smaller. The overview shows not enough recipes, even on large screens.

Important Features

Some features were mentioned again in the final discussion, which already appeared previously, or were highly emphasized by the participants. Based on this, these features appear to have a significant importance for the participants, and therefore might be of value to a future version of the *meSchup* environment.

Features for the *meSchup* Server:

Policy structure for rights management All of the participants agreed, that a policy structure to manage and restrict the privileges a recipe has, should be available if more users join the platform. This approach is described in section 3.5.13 and was previously mentioned during the first group discussion. The participants stated that they would be hesitant to deploy behaviour into their smart environment without knowing which aspects of their environment it could affect.

Store sensitive data externally Another feature already mentioned during the first discussion, is privacy related. Some approach should be taken to protect sensitive information in rules from being published. This approach was previously described in section 5.3.1.

Features for the Community Website:

Comments Being able to comment on recipes was already mentioned during the intermediate interviews and again emphasized in the final discussion. This was an expected result and should be taken into account for the future development of the platform. The participants stated the same use cases as described in section 3.5.2. Mostly, to inform the creator of a recipe about errors and propose new enhancements for it.

Versioning and updates Another feature previously mentioned during the intermediate interview is the possibility to apply updates and establish a versioning of recipes. This approach is described in section 3.5.12. It is safe to assume that not all recipes that are published will remain without the need for change. To cope with rework of existing recipes, an update functionality is essential. Therefore, this feature should also be taken into consideration for future development of the platform.

Additional Features

The discussion yielded a lot of feedback regarding the *meSchup* server as well as the community website. The most important ideas for features will be described in the following.

Features for the *meSchup* Server: There was a lot of feedback regarding the *meSchup* server and its functionality. Since evaluating this platform was not the main goal of the study, the following will show only an extract of the gathered feedback.

Automatic event filtering One characteristic of the *meSchup* platform is that whenever a sensor event is received, all rules are executed. Irrespective of the fact whether the corresponding data is used in the rule. Common practice to prevent a performance drawback is to check at the beginning of the rule whether the rule should be executed based on the current event. To simplify this approach for future users, an idea was proposed to automatically create these checks based on a user's input. In that case, the user would choose on which events to execute the rule, and on which to omit the execution.

Device view in editor When creating a rule, the user has to know, which devices are currently available and which modules are installed on them. Therefore, one user proposed the idea to show a visualization of all available devices next to the editor window. Each device would be displayed together with its currently equipped modules. This visualization helps the user to quickly find out the names of this devices as well as the options he has with them. Furthermore, it can be used to

provide an intuitive user interface to solve the problem of *automatic event filtering*. Therefore, the user would choose the appropriate modules he wants to receive sensor data from, and then mark them accordingly. For incoming data from all unmarked modules, the corresponding rule would not be executed.

Historic sensor values During the development of rules, users have to work with sensor data from their corresponding devices. To be able to write meaningful rules, a user might have to compare sensor readings from different states. For example, a rule that is based on a brightness sensor could be using sensor readings, which represent a high and a low brightness value. Retrieving these values can become difficult, if the sensor is in another room. The user would have to leave his computer and therefore cannot see the current value of a sensor. To solve this, all participants were in favour of a feature that lets them view historic sensor values. A visualization of all sensor readings received by the *meSchup* server within the last 60 seconds, would be highly appreciated.

Features for the Community Website: The participants noted that they were mainly engaged in writing rules and barely exchanged recipes with each other. Nevertheless, they proposed some additional features, which might be valuable for future users of the website.

Hardware visualization Similar to the proposed visualization of hardware in the editor view on a user's on-site setup, a visualization of the utilized hardware and its modules was suggested for the community website. Embedded in the recipe view, a user would then be able to see the types of devices used in the particular recipe, as well as the modules associated with these devices. With this, the user can distinguish whether this recipe matches his own set of hardware. Furthermore, he can judge the effect this recipe has on his smart environment, without even deploying it.

Missing hardware An extension of the previous feature is to show a list of devices and modules, which are missing for the recipe to work properly. This could be visualized based on the hardware visualization described before. Using the connection between the user's on-site setup and the community website, a list of available devices can easily be retrieved. Thus, determining the missing hardware is simple. Furthermore, the missing hardware is basically a "shopping list" for the user. The visualization could be enriched with the appropriate address, where a user could buy the missing parts. Due to that, every user would be able to quickly determine whether a recipe will work on his setup and be able to buy missing parts in case he intends to use it.

Compatibility report Another feature, which is closely related to the previous ones, is a compatibility report of a recipe. This feature should warn a user while deploying, in case his on-site setup does not cover all the devices, which are necessary to benefit from a recipe. This notification should contain detailed information about how to use the appropriate simulating feature of his setup. Due to that, the user would still be able to test the recipe, even without possessing all necessary devices.

Publish templates A feature proposed by one participant, was the introduction of templates regarding the descriptions of recipes. When publishing a recipe, the user is confronted with the possibility to describe it (see section 4.4.3). While the title and abstract of the recipe are obvious, the content of the description itself was unclear. Therefore, the participant stated that a set of predefined templates, including notes about the possible content of the description, would be useful. Those templates should provide a meaningful skeleton, for users to enter their text at the correct section. Furthermore, every user should also be able to define such templates himself and be able to use them in future recipes.

5.3.4 Quantitative Results

Due to the size of the study, it yielded only very few quantitative results. One participant had issues with the network connection between his on-site setup and the community website. Therefore, he could not use the built-in publish or deploy function that retrieves rules from his on site setup. Neither did he use the possibility to upload rules via a regular file upload.
The remaining two participants published three recipes in total. Every published recipe was comprised of one rule. Each rule had an average length of 39 lines of code.

5.4 Discussion

The qualitative feedback gathered through the user study is valuable and needs to be considered in detail. In the following, suggested features and the qualitative results of the study will be discussed.

5.4.1 Features

Proposals that were not yet discussed in section 3.5, are examined closer in the following.

Recipe Signing

To increase the security level of published recipes, the digital signing of recipes was proposed. The intention behind this approach is the introduction of authenticity as a security quality. In the case of the *meSchup* environment, this means ensuring that the user, who claims to be creator of a recipe, actually is the creator of it. In general, there are multiple ways of accomplishing digital signatures. Depending on performance requirements and usage, some will proof themselves more useful than others. For the sake of simplicity, in the following *PGP* is considered as a method for signing. Obviously, until a real deployment can be made, prototypes and benchmarks should be developed before deciding on the technology.

To include signing functionality, multiple challenges have to be overcome. Most importantly, every user that creates recipes has to possess the required means to use PGP. Among others, this mainly means having a private and public key pair. To simplify usability, these keys should be created when a user first initializes his on-site setup. This could automatically be done, so that the user needs no knowledge about the signing itself, or its inner mechanics. While the private key remains on the on-site setup, the public key can be integrated in any public key server available. Leaving the private key on the on-site setup is useful for signing while publishing recipes. This way, the signing of the corresponding data can be done directly on the setup without involving the client. In case a user has multiple setups, all should possess the same private key. Thereby, additional overhead of managing several keys for the same user can be avoided.

Implementation-wise, the community server would only serve digitally signed recipes. When publishing a recipe, the community server requests the corresponding rules in the same way, as if no signing was implemented. After receiving them, the community server would continue to follow the normal publishing process by parsing the rules and retrieving the corresponding devices. Once the device retrieval is finished, the community server would hash the complete recipe, i.e. including the descriptive meta information the user entered as well as device and rule data. This hash value, called *digest*, is sent to the user's on-site setup, encrypted with the private key, and sent back to the server. Upon a request of a recipe, the server responds with the corresponding recipe and the encrypted digest. The receiver (either a client e.g. web browser, or a on-site setup) can decrypt the digest using the creator's public key. Generating a digest of his own, he can compare both and check whether they are equal. In case they are equal, the decryption yielded the correct hash value and therefore the authenticity of the creator can be confirmed. In case of an inequality, the user can be notified that he might received a forged recipe. Every time a recipe is updated, for example by changing

the description or uploading a new rule, this procedure has to be repeated.

Whether this approach has good performance and scales well with reference to an increase in recipe creations and deployments, is difficult to tell without further benchmarks. Nevertheless, since only one additional data transfer is necessary between the community server and the creator's on-site setup (transferring the digest), it does not seem to be a major performance drawback. Calculating the hash value of a recipe has to be considered as well, yet hash calculations even for multiple rules and descriptive meta information should be in a sub-second range.

User Search

One missing feature, which was noted during the study was the possibility to search for users. This feature would enhance the search functionality to allow retrieving all recipes related to a specific user. Especially with the assumption in mind, that only a small fraction of end users actually create recipes, while others deploy them, this is an interesting feature. When a user deploys a recipe that closely fits his needs, he might be eager to find more recipes of the same creator, since there is an apparent overlap in use cases between both. This is closely related to the functionality of "following" a user, described in section 3.5.5.

On the implementation side, this does only need minor rework of the existing project. To be able to search for usernames, the *view* functions of the database require to be extended (see section 4.5.1). This is the only change, which is necessary to provide a search for users originating in the search field of the web interface. Another popular method of getting detailed information about a user, is clicking its username in the web interface (see fig. 4.7). To achieve this, an additional change in the project would be necessary. The click on a user's name in the detailed recipe view, should trigger a search with the username as a query. With both changes, the search for users would neatly be embedded in the current interface.

User Rating

Closely related to the search for specific users, is the feature that also allows to rate them. Trying to achieve higher security and protecting users from malicious recipes and creators, this could help finding users with a good reputation.

The implementation of this feature requires some modifications of the current system. On the one hand, some kind of representation needs to be added to the user interface that shows a user's rating. Furthermore, it should enable other users, preferably those, who deployed recipes created by him, to submit a rating of him. On the other hand, the backend of the community server needs to be modified to accept those ratings and persist them in the recipe's data structure.

5.4.2 General Feedback

This section summarises the feedback of the participants regarding various topics.

Device Handling

Based on the feedback of the participants, incompatibility of their on-site setup to a recipe they want to deploy seems to be the most pressing concern. This is affirmed by their feature proposals: *Hardware visualization*, *missing hardware* and the *compatibility report*. Since this affects both creators and users of the platform, significant effort needs to be taken to cope with this concern.
On the site of the *meSchup* platform, device handling could be improved as well. Especially in the final group discussion - at the time of which the participants had spent time programming rules - they stated ideas how to improve device and data handling during programming. Mainly the idea to attach a device view to the editor window seems like a promising starting point. This window should show the user information about the available devices and their current, as well as historical sensor readings. Since all participants agreed that they would have found such a feature useful and it does not require an extensive redesign, it should be considered for future development.

Security

A topic that appeared throughout the study was security. The participants stated that, for a real world application of the platform, they would want some kind of security system. Mainly, the policy management system was addressed as such (see section 3.5.13 and section 5.3.1). This would allow them to see what kind of behaviour any recipe is allowed to perform. Especially for users lacking the necessary skills to program or to understand the content of a rule quickly, this system would be helpful. Furthermore, the participants stated that in case of a recipe with more than a couple of dozen lines of code attached to it, they would probably not check the content themselves. Rather than checking the code they would prefer a system that could visualize and control the actions a recipe can

perform. While this feature seems like a promising idea to implement, such a feature should be considered during the design of a smart environment system. Implementing it in the existing *meSchup* platform would require significant effort. Therefore, further investigation into the topic to gather more information on different types of security systems that apply, should be done.

Publishing

When asked whether they would publish created rules not in a study, but in a real world application of the *meSchup* platform, all participants had a consistent opinion about the topic. They would publish their work, if they were certain, that other users would benefit from it, the functionality would not be trivial and the created rule would be reasonably robust. Whether other users would benefit from their work could be determined by examining, whether similar recipes already exist. Robustness could be analysed via the simulation feature by injecting a variety of events in order to test the behaviour in unrealistic, yet possible, scenarios.
All of the participants agreed, that they would not share rules, which contain sensitive information (e.g. passwords or keys). To cope with this issue, an information storage that lets users extract information (i.e. variables) from rules into a non-publishable space is needed. This way, all types of recipes can be shared, regardless of their usage and potential access on private accounts. Furthermore, a participant noted that he would be hesitant to publish a recipe accessing a (potential) electronic door lock. Even without a key or password in the rule, the fact that an electronic door lock is installed is sufficient to decide not to publish the recipe. However, if in case of a real application of the *meSchup* environment, the platform would not ask for a real name of its members, the participant would be less concerned about such sensitive information.

Deployment

Asking the participants whether they would deploy recipes they found on the community website in a real application of the *meSchup* platform, the answers were diverse. All of the participants agreed that, the community website is an important approach to gather functionality and to look for coding samples while creating own rules. Furthermore, all participants would also use the community website to get utility functionality that they can universally apply in their rules. When presented with the idea to embed searching capabilities directly in the on-site editor (as described in section 3.5.11), the participants agreed that this would simplify programming.
When deploying actual behaviour-related recipes, the opinions of the participants diverged. One participant stated that he would not deploy complete recipes for the

behaviour they implement. The participant would not want his smart environment to be controlled by a downloaded recipe. Nevertheless, the other participants stated that they were open to enrich their smart environment with additional recipes. Even more so, if the creator of a recipe or the recipe itself has a high popularity.

Therefore, to achieve high deployment rates (i.e. a lively community) the popularity and trustworthiness of creators should be a central concern of future development. In order to simplify the development process of rules, an efficient search for code samples could be integrated within the *meSchup* server.

5.5 Results regarding Research Questions

The following summarises the answers to the corresponding research question.

5.5.1 *Q1: Are users (creators) willing to share their creations?*

Section 5.4.2 shows that, in general, all participants were willing to share their work with others. Important for the future development of such platforms is, as described previously, enabling users to create robust solutions. Therefore, the simulation feature is a promising starting point of enabling users to write well-performing rules. In addition, users feel more motivated to share their work, if they know it would benefit others. Therefore, it is necessary to provide a possibility, where users can state their wishes, so that creators can get a better view on what the community could need.

5.5.2 *Q2: How do security and privacy aspects influence sharing and using of recipes?*

As described in section 5.4.2, security is a great concern of the users. They want to be informed about the actions a recipe is capable of performing, when deploying it in their smart environment. Furthermore, the users outlined their concerns about publishing sensitive information that is stored within their work. The former issue can be tackled by a policy system described in section 3.5.13. The latter, on the other hand, can be coped with by anonymous accounts as described in section 5.4.2 or an extraction of sensitive information in a non-publishable storage (see section 5.3.1).
Moreover, whether or not users deploy recipes relates to the trustworthiness and the

popularity of the recipe itself and its creator (see section 5.4.2).

Not handling these concerns, might lead users to be hesitating about deploying or creating new work.

6 Conclusion and Prospect

This chapter concludes the work of this thesis and gives an outlook for the future of smart environments as well as the sharing of their behaviour.

6.1 Conclusion

Based on the *meSchup* platform for rapid creations of smart environments, a web-based sharing mechanism was developed that lets users of such environments exchange their work and ideas. It incorporates functionality to annotate the created work with useful meta information for future users. Additionally, it offers the possibility to deploy new behaviour into an environment from the web-based interface. After developing the prototype an in-the-wild study was conducted. It yielded important feedback, confirming thoughts and issues that appeared throughout the development, as well as new ideas of coping with them.
The contribution of this work to the current state of the art is diverse. The web-based sharing of behaviour for smart environments is a novel approach in this area. In addition, the deployment strategy that lets users enrich their smart environments through a simple web interface is, also, new in the area of smart environments. To achieve this, a recipe structure was developed that combines easy deployment capabilities due to accumulated information about the behaviour and necessary information about its utilized hardware. Furthermore, it centralizes behaviour related information in a single data structure, which simplifies sharing and representation on the community website.
Impacts on the privacy of potential users received particular attention throughout the course of this work. Especially the thoughts about the policy management system was emphasized by the participants during the study. Preventing malicious behaviour from retrieving sensitive user data is one of the key challenges. Another challenge is to keep private data of users, for example passwords and keys, safe. This is especially important if a user creates smart behaviour locking and unlocking doors or accessing email accounts.

6.2 Open Issues

Some issues observed by the participants of the study, concern the overall usability and design of the community website. This was expected, since the main focus during the development of this prototype was functionality, instead of layout and design. Nevertheless, in order to make such a community website appealing for a large userbase, a modern and intuitive user interface is necessary.
Another issue reported by a participant was an instable network connection of the *meSchup* platform. Since some communication between the platform and the community website is necessary to use its publish an deploy functionality in full extent, this is an important issue to consider. Due to the fact that the platform is not a finished product and underwent some hardware changes previous to the study, this should be considered a teething issue.
Furthermore, there was an issue regarding the connection of a user's on-site setup and the community website, in the event of ungraceful terminations of it. After such an unplanned termination, the user's on-site setup could not connect to the community website again without further manual intervening. This is a problem of correct connection release.
The same is true for an issue that complicated the setup of a participant's smartphone. With additional realistic setups and studies, such incompatibilities can quickly be overcome.

Since the conducted study was relatively small and did solely comprise of software engineers, threats to the external validity cannot be ruled out. Therefore, additional studies are necessary regarding both, functionality of the *meSchup* platform in general (including the community website), as well as studies targeting end users with no programming background. This way, more realistic results can be gathered.

6.3 Future Work

Based on the work with the *meSchup* platform and the feedback that was gathered through the study, some features appear to be very promising future work.

6.3.1 Device Handling

As described in section 5.4.2, one of the biggest concerns of the participants is wanting to deploy a recipe without having the appropriate hardware. This needs to be taken into consideration for the future development of the community website. A suitable

visualization for the utilized hardware should be visible in the recipe view. Furthermore, it should clearly display which kind of hardware this recipes uses, which modules are attached to the corresponding devices and which of the hardware is present at a user's on-site setup. Furthermore, when a user deploys a recipe while not possessing all of the required hardware, additional means should be offered to test the recipe anyway. This could be realized with additional work on the simulation feature to incorporate both sensor and actuators in it. In that case, the user should be able to access detailed instructions on how to use the simulator to test a recipe before acquiring additional hardware.
Besides the visualization on the community website, the user's on-site server could be improved as well. As stated in the evaluation, a visualization of available hardware during the development of rules would be beneficial.

6.3.2 Publishing

As described in section 5.4.2, various work has to be done to motivate users to publish their creations. Mainly, an efficient and comprehensive search is necessary, so that users find appropriate recipes and can evaluate whether their own creations present novel functionality. Furthermore, a enhancement of the simulation features might not only motivate users to deploy and test recipes, but also engage them to test their own creations to make them more robust. Users feel more motivated to share their creations, if they can rely on them operating flawlessly in different environments.
Besides that, encouraging users to share privacy related recipes requires additional work as well. First of all, sensitive information from rules should be extracted in some kind of property space that is not publishable, yet accessible from within rules. Secondly, even in future versions of the community website, users should not be asked to supply their real names. This would prevent third parties from associating a user's name from his smart environment setup (e.g. electronic door locks or email accounts).

6.3.3 Deployment

Regarding deployment, some features should be implemented to motivate users to deploy recipes they find on the community website. As described in section 5.4.2, creators of rules and recipes can benefit from an efficient code search. The idea behind this search was described in section 3.5.11. Besides that, deployments of complete recipes might be increased by enhancing the rating system. Users feel less hesitant to deploy a recipe, if the author or the recipe itself is very popular. Both approaches can be implemented with reasonable effort.

6.3.4 Device Assignment

Another important feature, to enable users to run recipes deployed from the community website, is the device assignment. It should be a future functionality of the community website and the *meSchup* platform. With this feature the user, who deploys a recipe may assign, which of his devices represent the devices utilized in the recipe. This feature was also described earlier, in an outlook of possible functionality, please confer section 3.5.9 for this.
This would be a substantial functionality gain, since a user would no longer need to modify the deployed recipe for it to run properly.

6.3.5 Versioning and Updates

Another feature, which came up several times throughout the study (see section 5.3.3 and section 5.3.2) was the necessity to apply updates to recipes. With this, the creator of a recipe would be able to update the rules of it. The need for an update could be based on additional functionality or the removal of known errors. This feature was, described previously in section 3.5.12.

6.3.6 Security

As described in the evaluation (see section 5.4.2), security is an important topic to consider for future development. Especially when users without a technical background deploy recipes, a policy management system would be useful. The system was described in section 3.5.13. This way, users are informed about the capabilities of recipes. Due to that, security and privacy implications of deploying a recipe can be judged more easily. As mentioned earlier, the implementation of this feature would require significant rework and effort to be realized. Additional research about other types of security systems that might apply, should be done first.

6.3.7 Same Hardware Search

Another promising feature for future users, is a recipe search based on particular hardware. As described in section 3.5.4, if a user is able to search for recipes that make solely use of hardware he possesses, it might significantly lower the threshold of using the platform. Since the exchange of data between a user's setup and the community server is possible, an additional usability gain is within reach. Retrieving data about the

user's setup in order to find out what hardware he uses, could even automate the search for recipes with the same types of hardware.

6.3.8 User Interaction

Commenting is a feature, which was requested by the participants throughout the study (see section 5.3.2 and section 5.3.3). It is a necessary tool for users to exchange ideas and feedback among themselves. In order to achieve a lively community with evolving recipe sets and continuous feedback to provide better recipes, future development should include a commenting functionality.
Besides commenting, the participants repeatedly mentioned rating of recipes as well as the creators of them (see section 5.3.1, section 5.3.2 and section 5.4.1). On the one hand, establishing the possibility for users to judge the popularity of recipes, might motivate them to download these. On the other hand, receiving recommendations and popularity might also motivate the creators of recipes to expand their work and continue to improve it. Even more opportunities might appear when implementing additional features for the community interaction, as described in section 3.5.2. Therefore, a rating system or similar user interaction features should be part of the future development of the community website.

6.3.9 Larger Study

The conducted in-the-wild study was relatively small to be feasible within the limits of this thesis. A larger study containing more participants and a wider range of equipment should be conducted to get more realistic results. A significant part of the targeted user group contains people without a technical background. Therefore, the study should incorporate not only developers but pure users of the platform, to produce more realistic feedback. To achieve this, both the *meSchup* platform and especially the prototype of the community website should receive further work targeting the usability, robustness and feature-richness of the system.

6.4 Prospect

Smart environments appear to be a very promising future. With further advancement in technology, smart devices can become smaller, yet have more battery power and better sensors. Embedding those devices in our personal environments, makes our lives easier and more comfortable. More and more manufacturers produce devices that we can

interact with using wireless communication technologies and our smartphones. This ranges from smart-home automation like wall plugs, radiator controls or lighting to the control of complete entertainment systems. As stated previously in section 1.1, controlling such devices is tedious. Every manufacturer creates individual applications (e.g. for smartphones) or remote controls for his devices. These applications and remote controls are not compatible with other manufacturers, therefore a user tends to have one application on his phone for every smart device in his environment. To unify the interaction with these diverse devices a platform, like *meSchup* provides it, is necessary to build and simplify the handling of such a smart environment. Enabling users to share their ideas and creations as well as providing simple means to expand the functionality of one's smart environment, will further expand the targeted group of users. Therefore, creating a community platform should go along with the advancements of *meSchup* and smart environments platforms in general.

With the introduction of smartphones, more precisely the possibility for end users to write their own software for it, the technical world experienced a significant change. The distribution and variety of applications for this technology grew rapidly. Smart environments are at a similar point at the time of this thesis. The fundamentals of the technology are available and the spreading of smart environments can be supported with powerful technology platforms and a lively community.

Bibliography

[AO04] P. G. Argyroudis, D. O'Mahony. "Securing Communications in the Smart Home." In: ed. by L. T. Yang, M. Guo, G. R. Gao, N. K. Jha. Vol. 3207. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 891–902 (cit. on p. 23).

[Apa05] Apache. *CouchDB - A Database for the Web*. 2005. URL: http://couchdb.apache.org/ (cit. on p. 48).

[ARM14] ARM Ltd. *ARM mbed IoT Device Platform*. 2014. URL: https://www.mbed.com/en/ (cit. on p. 19).

[AVM15] AVM. *Full control with FRITZ!Box & Smart Home!* 2015. URL: http://en.avm.de/news/the-latest-news-from-fritz/2015/full-control-with-fritzbox-smart-home/ (cit. on pp. 9, 18).

[BB04] B. Bender, L. Blessing. "On the superiority of opportunistic design strategies during early embodiment design." In: *Proceedings of DESIGN 2004, the 8th International Design Conference*. 2004, pp. 1–6. URL: http://www.designsociety.org/publication/19742/on%7B%5C_%7Dthe%7B%5C_%7Dsuperiority%7B%5C_%7Dof%7B%5C_%7Dopportunistic%7B%5C_%7Ddesign%7B%5C_%7Dstrategies%7B%5C_%7Dduring%7B%5C_%7Dearly%7B%5C_%7Dembodiment%7B%5C_%7Ddesign (cit. on p. 13).

[BBH+14] T. Ball, S. Burckhardt, J. D. Halleux, J. Protzenko, N. Tillmann. "Beyond Open Source: The TouchDevelop Cloud-based Integrated Development Environment." In: (2014), p. 11 (cit. on p. 15).

[BGL+09] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, S. R. Klemmer. "Two studies of opportunistic programming." In: *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. New York, New York, USA: ACM Press, Apr. 2009, p. 1589. URL: http://dl.acm.org/citation.cfm?id=1518701.1518944 (cit. on pp. 14, 37).

[BGLK08] J. Brandt, P. J. Guo, J. Lewenstein, S. R. Klemmer. "Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice." In: *Proceedings of the 4th international workshop on End-user software engineering - WEUSE '08*. New York, New York, USA: ACM Press, May 2008, pp. 1–5. URL: http://dl.acm.org/citation.cfm?id=1370847.1370848 (cit. on p. 14).

[CD07] D. J. Cook, S. K. Das. "How smart are our environments? An updated look at the state of the art." In: *Pervasive and Mobile Computing* 3.2 (Mar. 2007), pp. 53–73. URL: http://www.sciencedirect.com/science/article/pii/S1574119206000642 (cit. on p. 17).

[Co08] S. T. Co. *Seeed Studio Bazaar, Boost ideas, Extend the Reach*. 2008. URL: http://www.seeedstudio.com (cit. on p. 22).

[DBN14] S. K. Datta, C. Bonnet, N. Nikaein. "An IoT gateway centric architecture to provide novel M2M services." In: *2014 IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, Mar. 2014, pp. 514–519. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6803221 (cit. on p. 18).

[DKR99] E. L. Deci, R. Koestner, R. M. Ryan. "A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation." In: *Psychological Bulletin* 125.6 (1999), pp. 627–668. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-2909.125.6.627 (cit. on p. 16).

[Git08] I. GitHub. *GitHub*. 2008. URL: https://github.com/ (cit. on p. 34).

[Hac13] Hackster.io. *Hackster.io - The community dedicated to learning hardware.* 2013. URL: https://www.hackster.io/ (cit. on p. 21).

[HDK08] B. Hartmann, S. Doorley, S. R. Klemmer. "Hacking, mashing, gluing: Understanding opportunistic design." English. In: *IEEE Pervasive Computing* 7.3 (July 2008), pp. 46–54 (cit. on p. 13).

[Hip05] E. von Hippel. "Democratizing innovation: The evolving phenomenon of user innovation." In: *Journal fur Betriebswirtschaft* 55.1 (Mar. 2005), pp. 63–78. URL: http://link.springer.com/10.1007/s11301-004-0002-8 (cit. on p. 13).

[Ins14] Instructables. *Instructables - DIY How to Make Instructions*. 2014. URL: http://www.instructables.com (cit. on p. 19).

[Joy09] Joyent Inc. *NodeJS*. 2009. URL: https://nodejs.org/en/ (cit. on p. 48).

[KS15] T. Kubitza, A. Schmidt. "Towards a Toolkit for the Rapid Creation of Smart Environments." In: vol. 9083. Springer International Publishing, 2015. Chap. Towards a, pp. 230–235 (cit. on p. 17).

[KSL03] G. von Krogh, S. Spaeth, K. R. Lakhani. “Community, joining, and specialization in open source software innovation: a case study.” In: *Research Policy* 32.7 (July 2003), pp. 1217–1241. URL: http://www.sciencedirect.com/science/article/pii/S0048733303000507 (cit. on p. 16).

[Lan01] M. Langheinrich. “Privacy by Design — Principles of Privacy-Aware Ubiquitous Systems.” In: ed. by G. D. Abowd, B. Brumitt, S. Shafer. Vol. 2201. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, Oct. 2001, pp. 273–291 (cit. on p. 24).

[Lin10] F. Lindesay. *Jade - Template Engine*. 2010. URL: http://jade-lang.com/ (cit. on p. 48).

[Luk13] J. Lukic. *Semantic UI*. 2013. URL: http://semantic-ui.com/ (cit. on p. 48).

[LW03] K. Lakhani, R. G. Wolf. “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects.” In: *SSRN Electronic Journal* (Sept. 2003). URL: http://papers.ssrn.com/abstract=443040 (cit. on pp. 16, 41).

[Nar93] B. A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT press, 1993 (cit. on p. 14).

[NVR+11] V. Nuhijevic, S. Vukosavljev, B. Radin, N. Teslic, M. Vucelja. “An intelligent home networking system.” In: *2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)*. IEEE, Sept. 2011, pp. 48–51. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6031824 (cit. on p. 18).

[NWET04] P. A. Nixon, W. Wagealla, C. English, S. Terzis. “Security, privacy and trust issues in smart environments.” In: *Smart Environments: Technologies, Protocols and Applications* (2004) (cit. on p. 23).

[PBL13] L. Ponzanelli, A. Bacchelli, M. Lanza. “Seahawk: Stack Overflow in the IDE.” In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, May 2013, pp. 1295–1298. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6606701 (cit. on pp. 15, 37).

[Tel] Telekom. *Was ist Smart Home?* URL: https://www.smarthome.de/was-ist-smarthome (cit. on pp. 9, 19).

[WAA+15] K. Wolf, E. Abdelhady, Y. Abdelrahman, T. Kubitza, A. Schmidt. “meSch – Tools for Interactive Exhibitions.” In: British Computer Society, July 2015, pp. 261–269. URL: http://dl.acm.org/citation.cfm?id=2835270.2835346 (cit. on pp. 10, 27).

All links were last followed on May 1, 2016.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature