VRIJE UNIVERSITEIT BRUSSEL

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

Mashup Tool for Distributed Physical-Digital User Interfaces

TIM LEENAERS

Academic year 2017 - 2018

Promoter: Prof. Dr. Beat Signer
Advisor: Audrey Sanctorum
Faculty of Science and Bioengineering

Afstudeer eindwerk ingediend in gedeeltelijke vervulling van de eisen
voor het behalen van de graad
Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Mashup Tool for Distributed Physical-Digital User Interfaces

TIM LEENAERS
Academiejaar 2017-2018

Promotor: Prof. Dr. Beat Signer
Advisor: Audrey Sanctorum
Faculteit Wetenschappen en Bio-ingenieurswetenschappen

Abstract

Over the years, the amount of data in our lives is increased drastically. Not only the Web, but also devices around us overwhelm us with data. To manage this flood of data, mashups have been created over the years. While first generation mashups were focussed on the Web, new technologies like Distributed User Interfaces (DUIs) and the Internet of Things (IoT) offered new opportunities for a new generation of mashup tools. Mashup tools for DUIs and IoT have been developed over the years, but none of them integrates both.

This thesis offers a solution, namely the creation of a mashup tool for Distributed Physical-Digital User Interfaces. Related work in the fields of mashup tools, DUIs and IoT is investigated. Afterwards a general solution is defined following an iterative design process. Web technologies are used to implement the final product. Flowcharts are being used to construct the mashups.

With the designed mashup tool, users are able to connect Web of Things (WoT) devices and several display devices (phone, tablets, computers) to the system and design Physical-Digital User Interfaces. A mashup which allows different display devices to control a WoT device, or get data from other physical devices are example applications.

A user study with unexperienced programmers showed that the system is easy to use and helped them create advanced applications. In addition, some limitations are described and possible solutions to overcome them in the future are given. Last but not least, some extensions that can be made to the system in the future are discussed.

Acknowledgements

First of all, I would like to thank both the promotor and the advisor of this thesis: Prof. Dr. Beat Signer and Audrey Sanctorum. Without their guidance and support, things would be a lot more difficult. Their feedback, together with those of fellow students during progress presentations, was of a great help for improving this thesis.

Next, I would like to thank all the other people that tested the developed program during multiple stages of the process. This tool was made for unexperienced developers, so their feedback and advice was of great value.

Furthermore, a word of thanks and appreciation for all the teachers I had during my life as a student, going from pre-school to high school, from the university college to this university. In the end, each of them had a small or big role in becoming who I am.

Finally, a big thank you to my parents, my family, all of my friends and my dog. If school was hard sometimes, coming home, meeting up or going for a walk were the ideal ways to keep going.

Contents

1 Introduction

Data is everywhere. Since the introduction of the Web, the amount of data is increased drastically. A lot of personal data is floating around, websites overwhelm us with articles, and since the introduction of the Internet of Things it is possible for physical things to share data and to be controlled over the Web. For a person, it becomes hard to manage all these fragmented data. Luckily, solutions have been proposed during the years.

As the Web 2.0 was introduced new functionalities became possible [55]. Third parties could be accessed by an API and data could be extracted. Users were then able to select data from multiple resources and build their own application with it. This technique is done by the use of mashups [36]. Early examples of mashups are Yahoo Pipes! [58], Microsoft Popfly [40] and Intel MashMaker [16]. All of them could scrape data from the internet and make new applications with it. With Microsoft Popfly [40] for example, it became possible to get all the photos of a Flickr account and show them in a carousel. With Yahoo Pipes!, users were able to get data from multiple online newspapers and create their own news site with it. Nonetheless, most of these mashups tools are not available anymore, or the development of them was stopped [7, 15, 64].

However, technology kept evolving and new opportunities for a new generation of mashup tools arose. One of them is the introduction of Distributed User Interfaces [71]. With DUIs, users are able to distribute their work over multiple screens, to share it to others, or work together in a group with multiple devices. This cross-device interaction became the subject of a new generation mashup tools that helped developers create distributed interfaces. Examples of such mashup tools are MultiMasher [31], Panelrama [76], XD-Studio [52], Weave [8] and DemoScript [9].

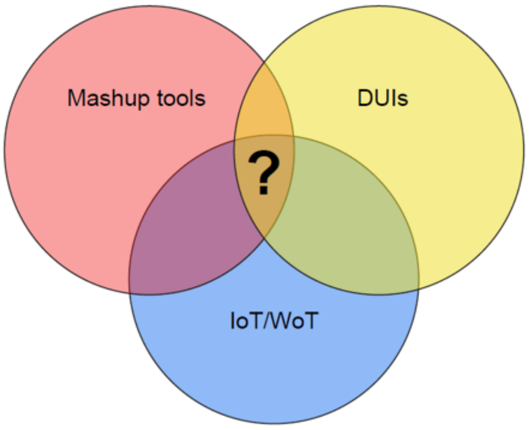


Figure 1.1: Mashup tools for DUIs and IoT exist, but limited research is done to develop a mashup tool for digital-physical user interfaces

Together with Distributed User Interfaces, the Internet of Things is a remarkable new technology. With IoT, physical devices become digitally available. Sensor data can be requested for over the Internet, lights can be turned on and off with mobile phones. New opportunities for different fields emerged [1]. Over the years, some mashup tools are developed in order to create applications that are able to control smart things. Open Sen.se [56], Node-RED[1], WoTKit [5] and glue.things [35] are some examples.

Until now, limited research is done on how to integrate DUIs and IoT/-WoT into a single mashup tool. The goal of this thesis will be to develop a mashup tool where it is possible to combine these two technologies. With this tool, users should be able to create, for example, an application where a smartwatch can be used to control a WoT lightbulb, a smartphone can change a map that is displayed on a tv screen, while sensor data shown on a tablet will be updated based on the new location of the map. Meaning the mashup tool should have the possibility to distribute data across multiple devices as well include information retrieved from smart things. In addition, these smart things should be controllable.

[1]https://node-red.org/

The goal is to make the tool as simple as possible where the end user should not know any programming language in order to use the tool. In this way, unexperienced developers should be able to create an advanced application, only by dragging and clicking around in the mashup tool.

In order to start with the development of this tool, existing work is investigated. First, related work in mashup tools, Distributed User Interfaces and the Internet of Thing (and Web of Things) is done separately. Next, research to overlapping fields is done. Mashup tools for DUIs will be examined, as well as mashup tools for IoT and WoT applications. Likewise, the combination of DUIs and the IoT and WoT is studied.

Based on the related work, a general solution will be proposed. Important guidelines from the literature reviews will be taken in mind and integrated in the final solution. Personas and a scenario are defined in order to test the program afterwards. A User Interface Design Process is performed before actually creating the application. User Class Descriptions, Usability Requirements, Tasks Modeling, User Object Modeling, a Style Guide and finally Prototyping are part of this process.

Done testing the prototypes, the final mashup tool is implemented. The Implementation chapter will provide all the technical details about how the tool is developed. A User Study is performed in order to evaluate the newly constructed mashup tool. Finally, Future Work is discussed and a Conclusion about this thesis given in the final chapter of this thesis.

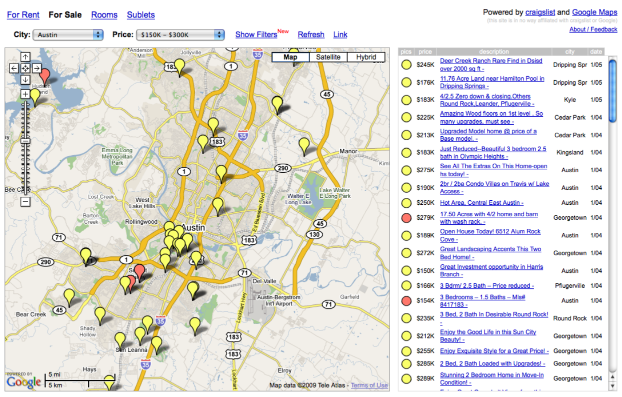
2
Related Work

In this chapter, the domain of mashup tools, Distributed User Interfaces (DUIs) and the Internet of Things (IoT) will be investigated. First, all the different topics are considered separately. Afterwards, work in overlapping fields is discussed.

2.1 Mashup Tools

The term 'mashup' originally comes from the music scene. It is derived from mixing songs to create a new song, regardless of the sources [28]. Nowadays, mashups are present in multiple domains like digital, video and web. In the paper "Elucidating the Mashup Hype: Definition, Challenges, Methodical Guide and Tools for Mashups" [36], the authors propose the following definition for the term mashup: "*a mashup is a web-based application that is created by combining and processing on-line third party resources, that contribute with data, presentation or functionality*".

The major breakthrough for mashups occurred when the Web 2.0 was introduced and new functionalities became possible [54]. Computers could now get dynamic access to data by using AJAX, APIs are used in order to share data. The web is now interactive, where it is possible for users to not only download, but also upload content. A lot of data is available and can be used by multiple parties [49]. This enabled new ways of creating mashups, with the consequence that a lot of them were created since the introduction of the term Web 2.0 back in 2004 [2]. A good example is housingmaps.com[1], which was the very first Google Maps mashup. On this site, houses that are on Craigslist are shown on Google Maps. It is said to be the first Web 2.0 application [62]. An example of housingmaps is shown in Figure 2.1. At the top of the page, users can choose what kind of accommodation they want to see (e.g. houses for rent of for sale). Next, some filters can be applied. Users can select a city they want to live in, together with a price range. Once these filters are applied, the results are shown in a list on the right side of the page and on a map on the left side.



City: Austin
Price: $150K - $300K

Powered by craigslist and Google Maps
(this site is in no way affiliated with craigslist or Google)

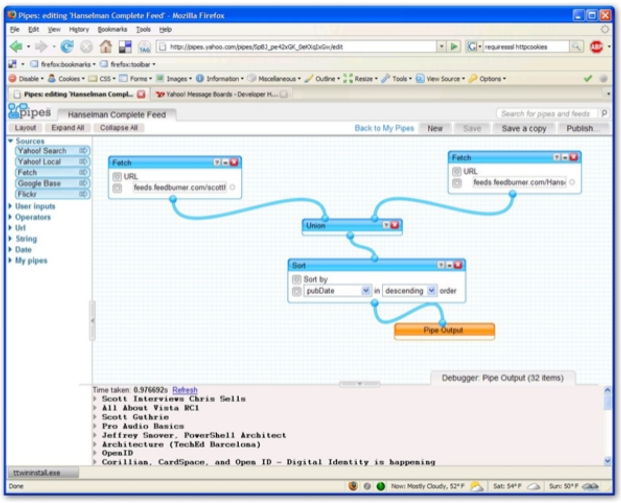
pics	price	description	city	date
🟡	$245K	Deer Creek Ranch Rare Find in Dsisd over 2000 sq ft -	Dripping Spr	1/05
🟡	$176K	11.76 Acre Land near Hamilton Pool in Dripping Springs -	Dripping Spr	1/05
🟡	$183K	4/2.5 Zero down & closing.Others Round Rock,Leander, Pflugerville -	Kyle	1/05
🟡	$225K	Amazing Wood floors on 1st level . So many upgrades, must see -	Cedar Park	1/04
🟡	$213K	Upgraded Model home @ price of a Base model. -	Cedar Park	1/04
🟡	$183K	Just Reduced--Beautiful 3 bedroom 2.5 bath in Olympic Heights -	Kingsland	1/04
🟡	$275K	See All The Extras On This Home-open hs today! -	Austin	1/04
🟡	$190K	2br / 2ba Condo Villas on Travis w/ Lake Access -	Austin	1/04
🟡	$250K	Hot Area, Central East Austin -	Austin	1/04
🔴	$279K	17.50 Acres with 4/2 home and barn with wash rack. -	Georgetown	1/04
🟡	$189K	Open House Today! 6512 Alum Rock Cove -	Austin	1/04
🟡	$272K	Great Landscaping Accents This Two Bed Home! -	Georgetown	1/04
🟡	$150K	Great Investment opportunity in Harris Branch -	Austin	1/04
🟡	$166K	3 Bdrm/ 2.5 Bath -- Price reduced -	Pflugerville	1/04
🔴	$154K	3 Bedrooms -- 1.5 Baths -- Mls# 8417183 -	Austin	1/04
🟡	$235K	3 Bed, 2 Bath In Desirable Round Rock! -	Round Rock	1/04
🟡	$212K	Enjoy the Good Life in this Sun City Beauty! -	Georgetown	1/04
🟡	$255K	Enjoy Exquisite Style for a Great Price! -	Georgetown	1/04
🟡	$285K	2 Bed, 2 Bath Loaded with Upgrades! -	Georgetown	1/04
🟡	$289K	Stunning 2 Bedroom Home in Move-In Condition! -	Georgetown	1/04

Figure 2.1: Example of housingsmaps.com: Houses on the right that match the filters are marked on the map

Murthy [48] defines the process of creating a mashup as follows: first, we obtain data from different sources, then combine them in a new application. We do this by a process that consists of three parts: data extraction, data matching (the transformation of data in the expected format for an endpoint) and data integration. With the introduction of these techniques, new challenges arose. Multiple unique mashups can be made, and it is not feasible to only let expert developers create those mashups. Together with these experienced users, an average Internet user should be able to create a mashup. But for them it may be hard to just create the desired mashup at once. Creating mashups namely requires the knowledge of programming languages like PHP, JavaScript and HTML [79]. Maybe, it would be better to serve a set of ready services to a user (a toolkit) with which the user can experiment and create their own content. It might be challenging to develop such a toolkit, but it the end, it can pay off [10]. This is an example of a

[1]`http://www.housingmaps.com/`

user-driven product evaluation where users get a set of tools and build products on their own [67]. In the past years, multiple of those mashup tools were developed in order to help users construct their own mashups [61]. One example of a mashup tool is Yahoo! Pipes [58]. The author of the book 'Yahoo! Pipes' describes that the Internet is our new database, and we need a tool to merge, sort, search and filter the data that is available on the web. Yahoo! Pipes focuses mainly on news feeds and lets their users create custom feeds. There are more than 24 modules that will help the user create his new web application. The user is able to just drag and drop some feeds on a canvas and combine them with some connectors. Users are able to save their pipes and share them with other users. An example of the interface can be found in Figure 2.2. Unfortunately, Yahoo stopped the development of Yahoo! Pipes in 2015 [64]. In the early years, a disadvantage of Yahoo! Pipes was the lack of presenting and understanding the RDF format. To overcome this, DERI Pipes [38] was created. DERI Pipes is built with Yahoo Pipes! in mind, but is able to understand RDF and output data in the RDF format. They call themselves a Semantic Web Pages engine (SWP) and enable the user to create semantically-enhanced mashups.

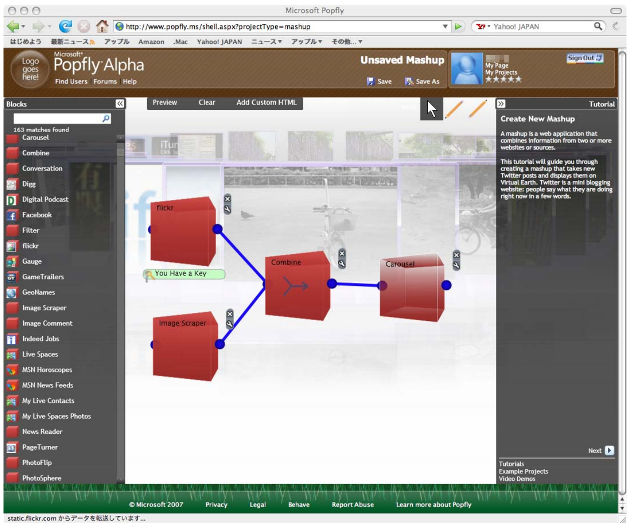

Pipes: editing 'Hanselman Complete Feed' - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://pipes.yahoo.com/pipes/Sp8J_pe42xGK_0eKXqIxGw/edit
Pipes: editing 'Hanselman Compl...
Yahoo! Message Boards - Developer H...
Pipes
Hanselman Complete Feed
Layout
Expand All
Collapse All
Back to My Pipes
New
Save
Save a copy
Publish...
Sources
Yahoo! Search
Yahoo! Local
Fetch
Google Base
Flickr
User inputs
Operators
Url
String
Date
My pipes
Fetch
URL
feeds.feedburner.com/scottl
Fetch
URL
feeds.feedburner.com/Hans
Union
Sort
Sort by
pubDate
in
descending
order
Pipe Output
Debugger: Pipe Output (32 items)
Done

```
Time taken: 0.976692s Refresh
▸ Scott Interviews Chris Sells
▸ All About Vista RC1
▸ Scott Guthrie
▸ Pro Audio Basics
▸ Jeffrey Snover, PowerShell Architect
▸ Architecture (TechEd Barcelona)
▸ OpenID
▸ Corillian, CardSpace, and Open ID - Digital Identity is happening
```

Figure 2.2: The Yahoo Pipes! interface

A drawback of Yahoo! Pipes and DERI Pipes is the absence of a visual representation of output data. The user is only able to view their results in plain text format. Mashmaker [16] is a mashup tool developed by Intel that allows users to visualise the collected data. Like the previous tools discussed, the main goal of Mashmaker is to allow non-expert users to easily create their own mashups. An example application can be the following: when a user wants to buy a house, they can add a housing website to Mashmaker. Once this is done, Mashmaker will create a data tree. When the user then clicks on a node (i.e. a house), Mashmaker will suggest widgets to apply to the node (e.g. look for shops nearby this house). These widgets can be edited by the users if they want to (e.g. specifying that 'near' means '1 kilometer'). The disadvantage of Intel Mashmaker is that it is only possible to visualise data, and not to store the data in a XML file for other usage. Just like Yahoo Pipes!, Mashmaker is not available on the Internet anymore [15].

Microsoft Popfly [40] is a mashup tool that combines the best of both worlds and makes it possible for the user to visualise their results, but they also store output data in text formats (e.g. XML). It is said that the end user does not need any skill of programming languages to use Microsoft Popfly, but does need a little bit understanding of data formats. Blocks need to be connected in order to build a successful web application. An example application can be to select all pictures of a Flickr profile and show them in a carousel (see Figure 2.3). In this way it differs from the previous tools. A common point is the ability to store the created web application online so other users are able to reuse the created program and, if they want, edit it for their needs. Unfortunately, also Microsoft Popfly does not exist anymore these days [7].



Create New Mashup

A mashup is a web application that combines information from two or more websites or sources.

This tutorial will guide you through creating a mashup that takes new Twitter posts and displays them on Virtual Earth. Twitter is a mini blogging website: people say what they are doing right now in a few words.

Figure 2.3: Microsoft Popfly: easily combine a Flickr account and an image scraper to show images in a carousel

Depending on the proposed classification of mashups by the authors of the paper Elucidating the Mashup Hype: Definition, Challenges, Methodical

System	Main features	Type
Yahoo! Pipes [58]	- Create mashups from news feeds - Output during design time - Control features like loop, union, filter	flow
Intel Mash-Maker [16]	- Helpful widgets to filter results - Visual representation of output - Suggests mashups to users	flow
Microsoft Popfly [40]	- Create mashups from different services - Visual and textual representation of output - Share created mashup with other users	flow
Dapper [21]	- Highlight and extract text from any web page - Create API based on any website - Share data with other users	extract
Marmite [74]	- Combine existing web content and service - Visual and textual representation of output - Template data-flows available	extract

Table 2.1: Overview of well-known mashup tools

Guide and Tools for Mashups [36], the tools that currently have been discussed are *flow mashups*. Here, users customise the resource flow of a web page by combining multiple resources from different sources. Afterwards, these resources are transformed and integrated in the mashup application. The other kind of mashups the authors define, are the *extraction mashups*. These mashups are data wrappers that collect and analyze resources from different sources and merge them in one content page. As an example they give Dapper [21], a mashup tool that (just like the others) is not in use anymore. With Dapper, users were able to highlight content of a website that they wanted to extract. This data could be outputted in the desired format and shared with other users afterwards. Dapper lets users create an API based on any website. When creating Marmite [74], Wong and Hong had a look at Dapper and discovered that it is more a tool for programmers who can make use of the output Dapper produces: users had to be able to transform the generated XML file in a visual representation. With Marmite, users can add multiple steps and choose to see the result in a spreadsheet format, to visualise the results on a map, to directly insert in a database, or to make a combination of two or more. The interface of Marmite exists of 3 columns. In the first column, a number of operations are listed. Users can drag one or more operations to the second column and combine them with pipes to create a data flow. The operations can extract and process data from web pages. Output results are shown in the third column in the chosen format(s). If multiple formats are chosen, Marmite will generate a list box in this third column where the user can choose which format he wants to see. For example, the user can create a data flow where the results first will be outputted in a spreadsheet format, and afterwards will be shown on a map. In the output column, a list box with options 'spreadsheet' and 'map' will be generated in order to choose the format in which the results will be shown.

As a conclusion of this part, we can say that the *first generation mashup tools* are not that successful anymore given the fact that a lot of products are discontinued. However, new technologies were introduced, together with new possibilities to create *next generation mashups*. In Table 2.1, an overview of some mashup tools is given. In the following sections, two of these new technologies (Distributed User Interfaces and the Internet of Things) are discussed. Afterwards, some examples of mashup tools that were created to support the development of mashing up DUI and IoT applications are given.

2.2 Distributed User Interfaces

Distributed User Interfaces (DUIs) are user interfaces that try to surpass the classic user interfaces that only can be manipulated by one user at the time on one screen. They enable the users to distribute a UI element on their screen to one or many other screens at design and/or runtime [71]. A new environment is created when multiple displays and devices are connected to each other. Key factors of such a new system are the scale (that depends on the biggest device in the environment) and the amount of individual possibilities for users [66]. The creation of DUIs gained interest because of a cost reduction of digital displays and can nowadays be found in indoor and outdoor environments [19]. The introduction of new communication technologies like web services gave the opportunity to fully utilize the potential of interconnected mobile devices to create a distributed interaction space [41].

To come up with a general definition of a Distributed User Interface, Vanderdonckt considers the definition of distributed computing, which is "*a distributed system that consists of multiple autonomous computers that communicate through a computer network*" [69]. He took this definition as a basis for the definition of a UI distribution, which "*concerns the repartition of one or many elements from one or many user interfaces in order to support one or many users to carry out one or many tasks on one or many domains in one or many contexts, each context of use consisting of users, platforms and environments*". Distributed User Interfaces thus allow for a UI to be divided over a set of devices and displays. They take advantage of these multiple devices and displays instead of relying on just one of them [3]. Multiple forms of distributed interfaces exist and define a set of characteristics for these DUIs [45]. The following list gives an overview of different DUIs and their characteristics that have been investigated by other researchers:

- *Multiple monitors on the same computing platform can be used by one user.* In their paper, Bi and Balakrishnan [4] did a week-long study and compared the use of a small monitor and the use of a bigger screen (with more space for different windows and tasks) by a user. Results showed that users unanimously preferred the use of a large display. They thought it was easy and less time consuming not have to switch continuously between windows when completing a task.

- *Multiple platforms, that are in sync, can be used by one user.* In [32] the authors did a study to investigate the difference between a single monitor user and a multi monitor user. They did this in order to know if new design principles should be developed for multi monitor users. They logged the window management activity of both single and multi monitor users. Results of the tests have shown that multi monitor users tend to use a taskbar less and do more window interactions to switch between windows. Different windows were also longer visible to the users as they did not have the need to close or hide them to perform another task.

- *Information belonging to different users can be shared over multiple platforms.* Already in 1998, Dewan and Shen [12] developed a multiuser framework to support access control in multiuser interfaces. It should be possible to insert data in a database and access it on another device. The framework supported a set of protected objects like sessions, windows and hierarchal active variables. It is designed to control shared operations and add policies in a flexible way.

- *Some information can be held private on the personal screen of the user.* As discussed above, not everything should be shared and some things should only be accessible by the user specifically. Users have to be able to share interaction components with other users, but this has to require negotiation of interaction resources in a dynamic way [3].

- *Information can be moved between displays on a single platform.* In [43] a virtual toolkit is proposed in order to create virtual applications in a single user space with multiple displays available. In their example application, they show that it has to be possible to control the lighting and heating (physical UIs) in a room by a computer, and share this data with other displays in the room.

- *Partitioning tasks across different displays for a single user.* WinCuts [65] was a program where users were able to cut pieces out of a page and load it in new, smaller windows. These smaller windows could afterwards be distributed to other displays. Next to this, it was also possible for multiple users to share a cut of their screen to a central shared screen. This situation is depicted in Figure 2.4.

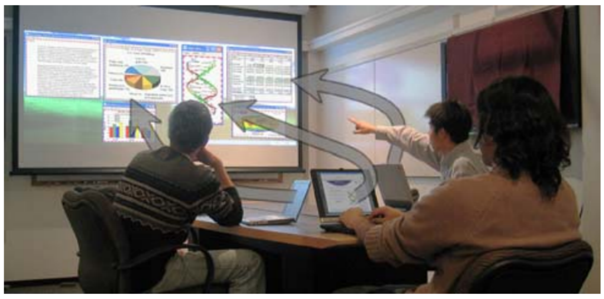


Figure 2.4: Users share information from their laptop by using WinCuts

A good example of a DUI where a single user can distribute multiple UI elements, is the 'Attach me, Detach me' program from Grolaux, Vanderdonckt and Van Roy [22] where it was possible for one single user to choose on which device and display certain UI elements should be available in order to accomplish a given interaction task in a better way. An example application they give is the painter's palette: the user can use one screen to make a painting, and another one to place his virtual color palette on to choose other colors. Another example is a digital version of the game Pictionary. Multiple smaller devices and a shared large device can be used to play the game. To prevent cheating, the toolkit provides some distinctions between users [44]. Further, Sjolund, Larsson and Berglund [63] created a remote GUI control on their smartphone that controls a Windows Media Player that is displayed on their TV. Users are able to play, stop and pause music numbers with their smartphone that are playing on TV.

Over time, mashup tools were created to help non-programmers create DUIs. Examples of these mashup tools are given later in this chapter. Next, the Internet of Things (IoT) will be discussed. With the introduction of IoT, new opportunities for Distributed User Interfaces became possible [37] and will help the world to work in a more optimal way.

2.3 The Internet of Things

The Internet of Things is a collective term for various aspects that connects the digital world with the physical world. A whole new class of applications and services will be enabled due to new developments in information and communication technologies [46]. The term IoT was introduced almost 20 years ago. The AutoID Labs at MIT were creating some RFID infrastructures at that time. In general, this is seen as the start of the Internet of Things [1]. Since then, new technologies have come along and so the scope IoT is far extended beyond RFID [75]. Nowadays we have phones, sensors, NFC tags, infrared tags, etc. that will be able to interact with each other to reach common goals [20].

IoT is characterized by the combination of physical and digital components that let us create new business models [77]. New opportunities for different industries will arise and the competition between competitors will change [57]. In their paper 'The Internet of Things: a Survey', Atzori et al. [1] listed some domains where they think the Internet of Things will play a big role: transportation and logistics, assisted driving, mobile ticketing, monitoring environmental parameters, tracking people's health, industrial plans, smart museums and gyms, the list is endless. Figure 2.5 shows an overview of multiple domains that will benefit of the IoT. Now, several years later, a lot of the proposed solutions are present in our daily lives.

However, there is one problem: the Internet of Things is more focused on the lower layers of the networking stack and much less on how to facilitate the development of new applications. In other words: the main attention goes to how transmit data between different actors and not how data can be collected, processed and visualised. The reason for this is that the early IoT systems were designed to operate in isolation, with the consequence that the IoT is a fragmented world. A lot of protocols have been proposed in the last decade, but none of them has made it to be the standard protocol for developing the Internet of Things. The authors of [25] therefore proposed the Web of Things (WoT). With WoT, known methods that are used on the Internet like REST APIs are used to allow communication between devices. IoT uses layers 1 to 6 from the OSI model, WoT only uses layer 7 (the application layer). With WoT, users can create programs, integrate data and services, deploy and maintain large systems more easily. IoT is ideal for hard-wired solutions and are more lightweight and optimized for embedded devices. IoT gives the everyday device an IP address and makes them interconnected with the Internet, WoT enables the devices to speak the same language [70].

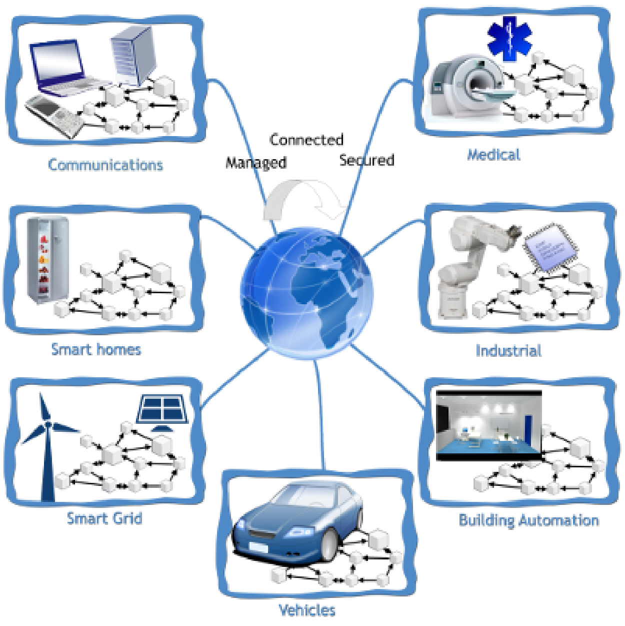


Figure 2.5: Interesting domains for the Internet of Things

2.3.1 The Web of Things

All things that are accessible by the Internet are called a resource. Each resource has a unique identifier (a URI) that is needed to be accessed by other devices and share its state with them. All of this can be done if we put things to REST. The main idea is to design applications which implement their functionality completely by defining a set of URI-addressable resources. HTTP is used as the access method for interaction between the resources. The big advantage is that each thing can be accessed like a web resource. In this way, things can be reused in different contexts and applications [73]. This approach of integrating things into the Web, is described by O'Sullivan and Igoe [53] as Physical Computing.

REST is nor a technology, nor a standard, but has an architectural style that defines how to use HTTP as an application protocol. Services are based on the HTTP protocol itself [17]. HTTP methods GET, POST, PUT and DELETE can be used to manipulate resources [33]. With REST, smart things can be linked together. The interaction between smart things can almost entirely happen from a browser, a tool that most users are familiar with and know how to use [34]. The resources can be linked and bookmarked and the results will be directly visible in the browser [26].

In their paper 'Towards the Web of Things: Web Mashups for Embedded Devices', Guinard and Trifa [24] list two alternative methods to enable REST-based interaction with embedded devices. In the first method, the device is directly made part of the web. When more and more devices become IP-enabled and have embedded HTTP servers, they can share their RESTful APIs on the web and are directly integrated. The second method works with Smart Gateways: this is an element in the middle that will function as a bridge between a device that does not talk IP and the web. Each gateway will have an IP address in order to communicate with the web and understands the protocols of the different devices that are connected to it. The first option seems to be the more promising one for the future and will be used during this thesis.

Linking physical objects with the web is known for several years. An early example is the linking a physical token (like a barcode) to a webpage [72]. In this way, users are directed to that page and can see more info about the product . Later, when new technologies were available, tiny web services could be integrated into most devices [13]. A famous example is the Cooltown Project of Intel where URIs and pages were associated to people, places and things. Infrared tags in the environment could be scanned in order to gain information [55]. The SenseWeb project is a platform that let people share their sensory readings using web services to transmit data to a central server [23]. However, all these approaches only have the ability to push data because the actors are considered to be passive. The Web of Things makes it possible to not only let actors push, but also retrieve data and to act on it [27].

Previously mentioned authors Guinard and Trifa founded a new company, called EVRYTHNG [2], that specialises in bringing the Internet of Things to REST. Their believe is that every physical thing can come alive digitally in some shape or form. In their book [25], they list some domains that will take of advantage of the Web of Things:

- Wearables: integrating them on the web so that the data is directly accessible by other devices and applications. With all this data available on the web, it should be much easier to develop new applications for health, fitness or elder care. It also ensures not to have a separate app for each of them.
- Smart homes and buildings: the problem of this domain are the many standards and protocols that are present to connect things to a network. With their company EVRYTHNG, they already created a smart home with smart products of different manufacturers, but they managed to let them talk to each other by using the Web of Things.

[2]https://evrythng.com/

- Smart cities: using web standards makes it easy to share sensor data with the public. Developers can use this real-time data and integrate it in various applications like traffic information on maps.
- Industry 4.0: all elements in a company can be connected due to the WoT. Companies can easier adapt to changing environments and actors can automatically decide how the best performance can be done, thanks to stored data.
- Marketing 2.0: digital content can be attached to products in order to make the product more alive. At EVRYTHNG, they worked together with a whiskey supplier: people could order a bottle of whiskey for Father's Day and write a special message which was then transformed to a QR code that was placed on the bottle. Fathers could later scan this code with their mobile phone and read the message of their children.

2.4 Mashup Tools Integrating DUIs and IoT

2.4.1 Internet of Things and Distributed User Interfaces

With the introduction of the Internet of Things and wearable devices, new opportunities for Distributed User Interfaces arose [37]. People can use their smartphone, laptop and smartwatch as an input device or output display to communicate with other devices [30]. In order to allow users to fully take advantage of all possible devices, they have to let them communicate with each other. It is important to combine the strengths of each device to build a multi-device ecosystem [11]. Examples of smartphones or smartwatches acting as a remote control for a TV are well known [59]. A famous example is Google Chromecast [3] where users can connect a small device to their TV (see Figure 2.6), and use their smartphone or laptop to send media to it. In addition, controlling this media (e.g. pause a video) and playing games with your smartphone as a controller are possibilities [14].

Likewise, multiple examples in the sports industry exist. Runkeeper[4] and Strava[5] are systems that keep track of people's positions and heart rates during physical activities using the built-in sensors and trackers in a smartwatch or smartphone. Additional functions are present to share live data with others, who can follow the user running or riding on their own tablet or computer. Coaches can follow if their athletes fulfill the required training

[3]https://store.google.com/product/chromecast_2015
[4]https://www.runkeeper.com
[5]https://www.strava.com

program (e.g. running 5 minutes at a high heart rate, afterwards cool down for 10 minutes at a low heart rate).

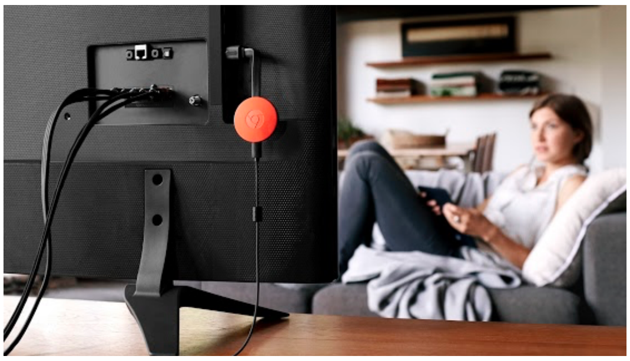


Figure 2.6: Google Chromecast (orange) connected to a TV using HDMI

Many more examples of controlling smart things with smart devices exist in multiple domains. People are able to control the thermostat in their house by using their computer, or another smart device that is present [60]. IoT and DUIs are loosely coupled these days and begin to play an important role in both people's professional and private life. However, not all people have programming skills to create their own applications. In order to overcome this problem, multiple mashup tools and toolkits for creating DUI and IoT applications were developed over time.

2.4.2 Mashup Tools for Distributed User Interfaces

Several toolkits are developed in order to help programmers develop new cross-device applications. In this sections, the following toolkits will be investigated: MultiMasher [31], Panelrama [76], XD-Studio [52], XD-Browser [51], a framework by Frosini and Paterno [18], Tandem Browsing Toolkit [29], Weave [8] and DemoScript [9]. An overview of these tools is given in Table 2.2

MultiMasher [31] presents itself as a visual tool for multi-device mashups. The goal is to build a tool to facilitate the quick and easy creation of web content of multiple sources, without the need for programming. Devices need to be connected to the MultiMasher server. Afterwards, the user can load the websites they want to mashup, select UI elements of that website and distribute these to the connected devices. The main limitation of MultiMasher occurs when interactions generate a list of updates to a database, for example when a comment is posted on one device, the submit button is triggered also on all the other devices. Another disadvantage is that sites that require authentication like Facebook and Twitter, cannot be easily integrated in a mashups. It is also not possible to design mashups for devices that are not connected to the system at runtime. Finally, the filesystem of users cannot be used.

The Tandem Browsing Toolkit [29] is similar to MultiMasher in the sense that multiple devices and multiple users are supported. However, like MultiMasher, Tandem Browsing Toolkit does not have the ability to access the filesystem of different users. It is more a tool to give the users privacy in a situation where a shared display is used. A login screen can be shown on a mobile screen of the user, while non-private information is shown on a public display. The Tandem also allows for improved communication within a group of users, which can simultaneously manipulate pages. A drawback of MultiMasher and Tandem Browsing Toolkit is the need for a fixed server. Frosini and Paterno [18] developed a framework where no fixed server is needed, distribution can be done across a dynamic set of devices using peer-to-peer communication. Multiple user roles can be created and multiple kinds of devices can be used. Distribution updates are processed by taking into account these roles and devices. As an example application, they present a giant screen with a QR code that can be scanned in by tourists by their mobile phone or tablet. Once they are connected to the program, they see a presentation on their screens that can be manipulated by the guide.

The previous toolkits all focus on multiple users using their own screen and sharing data to others and a central device. Panelrama [76] is a toolkit that is created for a single user, working with multiple devices. A user can divide a UI of an application into multiple panels. For each panel, a score can be added to show the importance of certain device characteristics to the usability of the panel. When used in a room with multiple devices, the panels can be distributed to the best-fit devices for an optimal experience. For example, a user can define a video on a page as one panel and assign to it that it is best fitted to a large screen. Afterwards, the video will be distributed to a large screen in the room. The key advantage of Panelrama is the use of existing web technologies to split down individual HTML elements, but it has also a disadvantage: with this technique, it is not possible to, for instance, split a video stream element in multiple parts and play all of these parts on different devices.

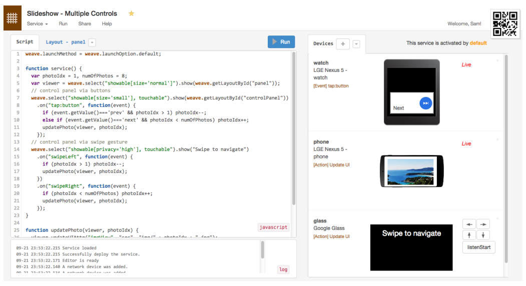
Another great advantage of Panelrama is the high control for the developers to automate UI distribution. This is something that is not available in Weave [8]. Weave allows developers to create cross-device wearable interaction by scripting. Thanks to a high-level JavaScript-based API, developers can distribute UI output and combine sensing events and user input across

System	Main features
MultiMasher [31]	- Administrator role for one user - One screen divided in zones. - Users send UI elements to shared screen
Tandem Browsing Toolkit [29]	- Give users privacy in certain situations - Multi-display and multi-user applications - Simultaneously manipulate pages in a group
Frosini and Paterno [18]	- Distribution across dynamic set of users - Different roles between users - No need for fixed server
Panelrama [76]	- For single user with multiple displays - Define UI elements in application - Show UI elements to best fitted devices
Weave [8]	- Emulating behavior of devices - Visualize JavaScript programming language - Expert programmers
XD Browser [51]	- Extend existing browsers - Parallel usage of multiple devices - Attention to unexperienced programmers

Table 2.2: Overview of mashup tools for Distributed User Interfaces

mobile and wearable devices. Cross-device behaviours can easily be tested on emulators or live devices. A drawback is the need to know a programming language (JavaScript) and less experienced users will not be able to create their own application. For expert developers, it is a good tool to test their code immediately and to simulate different input events. However, the developers of Weave found out that in this setting, it was difficult to see how the abstract application logic is executed with what devices and what kind of exact device behavior is mapped back to the code.

With DemoScript [9] they made an extension for Weave in order to answer the question how to bridge the gap between abstract programming logic and concrete UI behaviours. DemoScript visually illustrates a step-by-step execution of a selected portion of the entire program. It does this by generating automatically a cross-device storyboard. DemoScript analyses the program as the developer enters it, and generates the visualization in real-time. It can do this by understanding the underlying syntax and semantics of a cross-device framework. Like Weave, the disadvantage is that users need to know JavaScript in order to create applications. In Figure 2.7, a comparison between Weave and DemoScript is given.

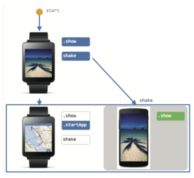


```
weave.launchMethod = weave.launchOption.default;

function service() {
  var photoIdx = 1, numOfPhotos = 8;
  var viewer = weave.select("showable[size='normal']").show(weave.getLayoutById("panel"));
  // control panel via buttons
  weave.select("showable[size='small'], touchable").show(weave.getLayoutById("controlPanel"))
    .on("tap:button", function(event) {
      if (event.getValue()==='prev' && photoIdx > 1) photoIdx--;
      else if (event.getValue()==='next' && photoIdx < numOfPhotos) photoIdx++;
      updatePhoto(viewer, photoIdx);
    });
  // control panel via swipe gesture
  weave.select("showable[privacy='high'], touchable").show("Swipe to navigate")
    .on("swipeLeft", function(event) {
      if (photoIdx > 1) photoIdx--;
      updatePhoto(viewer, photoIdx);
    })
    .on("swipeRight", function(event) {
      if (photoIdx < numOfPhotos) photoIdx++;
      updatePhoto(viewer, photoIdx);
    });
}

function updatePhoto(viewer, photoIdx) {
```

```
09-21 23:53:22.215 Service loaded
09-21 23:53:22.215 Successfully deploy the service.
09-21 23:53:22.171 Editor is ready
09-21 23:53:22.140 A network device was added.
```

(a) Weave

```
weave.select(':watch').show(photo)
  .on('shake', function(event) {
    weave.select(':phone')
      .show(photo);
    event.getDevice()
      .startApp('GoogleMaps', geo);
  });
```



(b) DemoScript

Figure 2.7: Weave (a) shows devices on the right side where developers can emulate actions. DemoScript (b) shows a storyboard, showing how devices react and how they are connected to each other

XDStudio [52] is a tool that is more suitable for users with a basic knowledge of web technologies. This program allows users to design for a multi-device environment on a single device by simulating other devices and their behavior, but also has the possibility to distribute the design process itself over multiple devices. The goal is to support the combined use of multiple devices for the same task and to support multiple users working on the same application. XDStudio uses techniques that are used in MultiMasher [31]: all parts of the DUI are continuously monitored and updated if needed. This is done by using a event replaying mechanism that sends local DOM events to the server, which sends the events back to all the clients involved.

Two years later, XDBrowser [51] was created with more attention to unexperienced programmers. This cross-device web browser was built in order to help non-technical users to adapt single-device web interfaces for cross-device use while viewing them in the browser. In their paper, they investigate how existing web browsers can be extended to support parallel usage of multiple devices. A drawback of the system is the ability to only work with pages that are served locally or stored offline. Many top sites forbid iframe embedding and browsers prevent cross-side scripting. A solution to overcome this problem is to work with proxy servers. Another disadvantage is the need to login on each device for sites that maintain a session. XDBrowser is still in its starting phase and is not that extensive. It only works well for a few interfaces like a mailing application and more research has to be done. The goal of XDBrowser is to reshuffle an existing interface over multiple devices, and less attention is given to users who want to reconfigure an interface on their own. In 2017, a new version XDBrowser2.0 [50] with a semi-automatic generation of cross-device interfaces was introduced.

2.4.3 Mashup Tools for IoT and WoT

With the introduction of the Internet of Things, new problems regarding creating mashups arose. Multiple sensors and devices with an internet connection (other than computers and smartphones) are all around us. It is said that in 2017, more or less 834 billion connected “things” will be in use. That is 31 percent more than in 2016. In total, the world will spend over 2 trillion dollars on these devices [68]. The biggest question is how are we going to let them work all together. One thing we should keep in mind, namely the main characteristics of mashup tools: simplicity, usability and ease of access [78]. Some examples like Clickscript [42], Open Sen.se [56], WotKit [5], glue.things [35] and Node-RED[6] are already in use.

Clickscript [42] is a Firefox plugin that lets the user create mashups by combining building blocks (like websites) with operations (e.g. a loop, if-else). A user should be able to, for example, create a program where every 10 seconds it checks the temperature (done by a sensor). If the temperature is higher than a defined threshold by the user, a cooler will be turned on. All of this can be done by combining building blocks. Clickscript is fully written in JavaScript and can easily access REST APIs using AJAX. Likewise, Open Sen.se [56] does similar work and uses REST APIs to connect with different

[6]https://nodered.org/

devices. It retrieves sensor data using HTTP and stores it. It allows for time scaling, averaging, summing, etc. Users can create a Sense Board where plugins can be used in order to visualise and process data. However, Open.se does focus more on the Internet of Everything, where Clickscript is designed for the Web of Things.

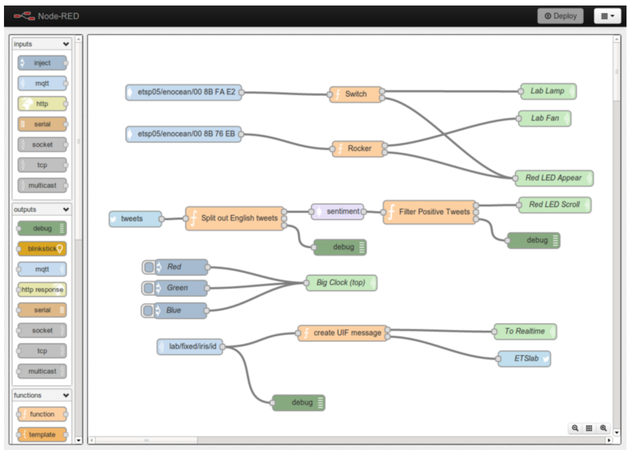


Figure 2.8: Node-RED interface

Node-RED is designed to create the Web of Things. It is similar to Clickscript and lets users create mashups by combining blocks. However, it reduces the need to write code and therefore is more user friendly. Devices, web services and software platforms are blocks that can be connected [35]. All flows are represented in the JSON notation and can easily be shared online. This cannot be done with Open Sen.se. An example of the interface over Node-RED is given in Figure 2.8.

Likewise, Open.se does not allow to share inserted devices with other on the platform itself. This is one problem that has been tackled by the developers of WotKiT [5]. WotKit has a sensor gallery (provided by the user himself and by the WotKit system). Users can generate and place widgets on a dashboard where all information will be shown. Users can create new sensor data from the ones they received on a management pipe. By means of a combination of modules and wires, a new pipe will be created. Users are then able to use this pipe to show data on the dashboard. A difference between WotkiT and Node-RED are the flows. The flows on different tabs or pages are not separated in Node-RED, there is only one flow for the entire system [6].

Glue.things [35] is a system that is built on Node-RED. It extends Node-RED with easy-to-use and predefined trigger and action nodes for devices and web services to create a mashup tool that focuses more on the user

System	Main features
ClickScript [42]	- Connect building blocks with operations - Easy access to RESTful APIs
Open Sen.se [56]	- Collections of plug-ins to use - Visualize and process data
Node-RED[7]	- Share mashups (stored in JSON format) - Browser-based flow editing
WotKit [5]	- Predefined trigger and action nodes - Publish and share device data streams
glue.things [35]	- Predefined trigger and action nodes - Publish and share device data streams

Table 2.3: Overview of mashup tools for IoT/WoT applications

and less on the developer. Glue.things works in the following way: first, users have to register their device at the Device Manager. Then, the Composer can be used to aggregate, manipulate and mashup device data streams in a visual way, using a web-based flow editor. It is this Composer that is built on Node-RED and makes it more easy for users to combine things. The Distribution Manager can be used to publish and share device data streams and distribute the mashup applications to customers. It is also possible to assign multiple users to one project and give them special roles. A summary of all the tools can be found in Table 2.3.

2.4.4 Conclusion

Multiple mashup tools exists, as well for creating Distributing User Interfaces (DUIs) as developing Internet of Things (IoT) applications. However, so far no mashup solution is present to combine both DUIs and IoT. This means that only expert developers can create distributed physical-digital applications and non-programmers are left out.

The goal is to create a mashup tool that will overcome this problem. The tool should be easy enough for end users without technical knowledge and therefore be less complex than the other mashup tools described, which still expect some knowledge of a programming language in some cases.

3 Solution

In this chapter, a solution to create a mashup tool for Distributed Physical-Digital User Interfaces is proposed. This solution should overcome the current problem, namely that no mashup tool exists that integrates both Distributed User Interfaces and the Internet of Things. First, a general description is given on how to solve the problem. In this section, the solution is compared with the related work discussed in the previous chapter. Some interesting guidelines and requirements that are mentioned in those papers and books which will be useful for this thesis will be described. Afterwards, some personas that will use the system are defined. These will play an important role in the scenario: an example of how the personas could use the system is given and will later on be used to test the system. Next, the design approach described by Moore and Redmond-Pyle in their book Graphical User Interface Design and Evaluation: A Practical Process [47] is followed. User classes will be defined and a list of usability requirements is given. Once the users and requirements are defined, a style guide can be made and an analysis about user tasks can be done. This will be helpful to start designing the UI and iteratively make changes to it.

3.1 General Description

To create a mashup tool for Physical-Digital User Interfaces, a combination of both Distributed User Interfaces and a Web of Things application should be made. For both topics, mashup tools are available as seen in the previous chapter, but none integrate both. However, it will not be easy to just alter an existing mashup tool to work for both a DUI and a WoT application. Both have different requirements and were not created with the intention to include things (in case of DUIs) or to show things differently on different displays (in case of WoT).

One thing that both have in common, is the separation of user levels. In most applications, there is an admin role (the creator of the mashup tool) and some user roles (the ones that use the application and do not have to know much about designing the application). Examples can be found in as well as in solutions for mashups tools for DUIs, like MultiMasher [31], for the Web of Things, like glue.things which is built on top of Node-RED [35]. For this thesis, a solution is proposed where there is one admin and multiple users. The admin can setup applications on one specific device. This device will act as a server and will receive all requests from different users (which will be display devices). These requests will be handled in the right way and the correct response will be sent back. This is the same approach as MultiMasher and the Tandem Browsing Toolkit [29], and most WoT mashup tools. Once an application is created, the administrator should be able to deploy it and share it with the right users.

It should be possible to distinguish multiple users. This can be done by assigning different display devices (also called users in this system) to different groups. In this way, a specification is possible where application A can only be shown on an iPad, while application B can only be used by users that are assigned to the group ‘Parents’. It thus should be possible to assign users to one ore more groups. However, it should also be possible to add and remove users individually to a list afterwards. An example situation can be that the groups ‘Parents’ and ‘iPads’ are selected for an application, but one iPad belongs to a child. The administrator should be able to delete this device from the participation list. Some applications require to be used by only one user at a time. This should also be specified upon the creation time of the app. An example application can be a music player, which only can be controlled by one user at a time.

Other requirements that came back in multiple papers were the need to add displays, devices and sensors to the system, the use of a flowchart to create applications, and the possibility the manage active users and applications. These requirements are explained in detail in the section ‘Define Users

& Usability Requirements' later in this chapter.

3.2 Guidelines to Create Web Things

One key to make the mashup tool easy to use for the administrator is an easy integration of Things. Thanks to the introduction of the Web of Things, it becomes much easier to communicate between different kind of physical devices. The book of Guinard and Trifa [25] contains a lot of good examples and guidelines to build the Web of Things. It is important that each device provides a REST API that is very clear for the users, so they know exactly what to do and what to get. Some of the most important guidelines given in the book are the following:

- *Use meaningful URLs.* Names with semantic value can be very helpful for the users. Do not use verbs in a URL, verbs are for HTTP methods. Use a plural form for aggregate resources, if a thing can do multiple actions, use 'actions' in the URL, and not just 'action'.

- *Web Things must expose their properties in a hierarchical structure.* In this way resources can easily be discovered. In addition HTML web pages can be generated to provide some more info about resources. Ideally the page www.example.com/pi/sensors should be accessible in json format and in HTML format. Using the HTML format, it is much easier for the user to read and discover more things about the resource. A list of all sensors can be given on this page so the user knows what he can request for.

- *Use JSON as the default representation.* The Things can support as many representations as they want, but should at least provide one in JSON at the minimum. It should also support UTF8 encoding for requests and responses. Object names in JSON payloads should be named using camelCase.

- *Web Things must support all the HTTP methods.* GET will be used to retrieve a resource, POST to create one, PUT to update and DELETE to remove an existing resource. A GET must be supported on the root URL so clients can receive information about the device.

- *HTTP status codes should be implemented.* The Web Thing should answer each request with a valid HTML response like 200 if the request was successful.

3.3 Guidelines to Create a DUI

One of the main goals of a distributed user interface is to make it easier for the user to perform tasks. With the use of multiple screens, productivity will increase with 10 to 30 percent [71]. In this system, it will be up to the administrator (the creator of the mashup) to show the right content on the right devices. The mashup tool itself should provide the creator all the tools they needs in order to perform his tasks successfully. The different kind of DUIs explained in [45] should be kept in mind. This means the following:

- *One user should be able to use multiple display devices.* The mashup tool should allow the possibility to create applications that can be distributed over multiple displays. An example application can be that the user uses his smartphone to search for a city, when afterwards all the necessary info is shown on a tablet.
- *Partitioning of tasks across different displays.* This is closely related to the previous one and the same example can be used. Multiple displays will make it easier for the user to perform certain tasks.
- *Information belonging to different users can be shared over multiple displays.* The mashup tool should allow users to work together to finish a task. As an example, the previous situation of searching for a city can be given, but now with different users owning different devices.
- *Information can be held private on the personal screen of the user.* The mashup tool should allow to build applications where some information is only been shown on the screen of the user. An example application can be that the user can scroll between a personal photo album and select a picture to be shared to a central screen that everyone can see.
- *Information can be moved between displays.* This is slightly different than the first point. In this case, the mashup tool should allow that a current state on a certain device of a user can be copied and placed on another device. If the user enters a text in a field, he should be able to copy the current state (including the current value of the field) and show it on another device.

3.4 Personas

Some personas are described to have a look on potential users of the system. This personas will later on play an important role in the scenario.

Jacob Daniels is a 21-year old who still lives at home with his parents and his little sister. Jacob studies Economics and has no programming experience. However, he knows how to use a computer and has basic knowledge of popular spreadsheet and presentation programs. He is interested in new technologies and owns a smartwatch, smartphone, tablet and laptop. He likes to use these smart things and is excited about new automation systems, but does not have time to fully understand the technology behind it and create one on his own.

Jane Daniels is the 19-year old sister of Jacob and studies Physics. Like Jacob, Jane has basic knowledge of computer programs, but has no interest in creating one on her own. Jane likes to travel and is looking for an easy way to see as much info about a city as possible. Sometimes Jane can be very lazy and refuses to stand up to turn on or off a light. People in the 21st century should be able to use their smart devices to do tasks like this, she thinks. Because of her laziness, she is not the person that is gonna spend much time on developing a system that is capable of doing so.

Ed Daniels is the 50-year old father of Jacob and Jane and works as an accountant. He uses some specialised programs to do his job, but does not know that much about computers. He owns some smart devices, but does not really have a clue how to use them. If it takes more than four steps to use a program, it is marked ‘difficult’ by him. Nonetheless, just like his son Jacob, he is interested in new technologies that change the world. He is prepared to invest in some smart things to use in the house, but does not want to pay external people to install all these things. His kids should be smart enough to make it work.

Rosa Daniels is the mother of the family and is a nurse. She does not have to use a computer often at work and does not care much about technology. She thinks it is all ‘too much’, but uses technology as it suits her. Since all her friends have a smartphone, she eventually also bought one. Currently she is against a smart home, but in the end she will adapt to the new situation and use her phone to control things.

3.5 Scenario

When all defined functional and non-functional requirements requirements are met and the program is ready to be used, the users should be able to create applications and use the system. To check if everything works well and is clear for the users, the following scenario should be completed.

1. Jacob starts up the system for the first time. No one has used it before, so Jacob has to enter a new password before entering the system. For

now on, every time someone wants to use the mashup tool and manage devices and users, the password should be given in order to enter the system.

2. Jacob creates some groups he can later on add devices to.

3. Jacob connects a few of his devices to the system. He takes his phone and tablet, passes the correct url and clicks on 'Connect' to send a request to the server. The administrator program pops up message that a particular device wants to connect to the system. Jacob accepts this device, gives it a name, and assigns it to different groups.

4. Jacob is not able to use any applications yet, since none are created at this moment. He uses the 'Disconnect' button on his smartphone and smartwatch to log off the system.

5. Jacob adds some WoT devices to the system:

   (a) A Raspberry Pi with a temperature and light sensor, and some LEDs.

   (b) A light bulb which can be controlled over wi-fi.

6. Jacob creates three different applications which can be used by the users:

   (a) The first application shows a map from Google Maps on a computer. Users with a smartphone are able to change the location and get some detailed information that is provided by Google Maps. Jacob specifies that only one smartphone user at a time can use this application. Tablet users can get more detailed information about the location (how to get there, the current weather, things to see, ...).

   (b) The second application makes it possible for all display devices to show some sensor data from the Raspberry Pi and from public sensors in one application. Jacob specifies the widths of each UI element in order to align it properly on different types of displays.

   (c) The third application involves only smartwatches and smartphones. Those devices should be able to control a light bulb. The appropriate text should be shown on the button to turn on/off the light on all devices.

7. Jacob is eager to test one of his programs. He takes his smartphone and tablet with him and connects again to the system. Since both devices were already accepted by the system in an earlier stage, they do not have to be accepted again and just see a list of applications they can use. On his smartphone, all three applications are listed. On his tablet, the third application to turn on and off the light is not available due to the passed settings.

8. Jacob fires up application 2 on his smartphone and tablet. On both devices he sees sensor data from the Raspberry Pi and third party sensors. The layout of the page is different on both devices, thanks to the settings that were defined whilst creating the application.

9. Jane sees her brother using the app and asks for the address to connect to the system. Jacob gives it and Jane connects to the system with her smartphone. Jacob now has to accept this new device and assign it to a group. When Jane gets accepted, she will not see any applications. Jacob has to add the device to various applications it can use. He gives the smartphone of Jane the permission to use all the applications.

10. Jane fires up the first application to search for some cities for her next trip in the summer. She asks Jacob to use his tablet, who gives it to her. On her phone she searches for 'Berlin, Germany'. The tablet shows more information about Berlin. Jacob tries to connect to the application with his smartphone, but gets rejected. He is happy that this happens, because he now knows his settings are working (only one smartphone user at a time can use Application 1).

11. The battery level of Jacob's tablet is very low and he wants to charge it. Jane has to connect her own tablet to the system if she want to work further. She does this and clicks the 'Distribute' button on top of the page on her smartphone. After doing this, she selects her tablet. On her brother's tablet, Application 2 will automatically be opened in the correct state.

12. Ed and Rosa want to see what their kids are doing. Ed is immediately interested and connects his computer to the system. After trying everything out, he gets Rosa to connect her smartphone to the system. Both are using Application 3 and are playing around by turning the light on and off.

13. Jacob wants to use Application 1, but cannot enter it because Jane is still using it with her smartphone. Jane rejects to close the application,

so Jacob kicks her out of the session by using the administration tool.

14. Jane is very stubborn and quickly connects again to Application 1 before Jacob even has a chance. Jacob takes desperate measures and not only kicks Jane out of the application, but also blocks from using it again.

15. Jane is very angry now and starts annoying the others by misusing the other applications. Jacob is tired of all this and blocks her completely from the system.

3.6 Define Users & Usability Requirements

This section will be used to describe the first steps in the design process described by Moore [47], namely the identification of the end users of the system. When doing this, it will become clear what the characteristics of the different users are. This will be helpful to specify usability requirements afterwards.

3.6.1 User Class Descriptions

Two user classes will play an important role in this system: Administrator and User. Administrators will create different applications, which can be used by various users (which own a display device to interact with the applications).

Administrator

The administrator is a human who has some knowledge of designing applications, without the need of an actual programming language. Some basic logical thinking and hints provided by the program should be enough to construct a simple application. As creating an app requires more knowledge than just using one, the minimum age to use this program will be a bit higher. For this system, the minimum age is set at 15. The administrator has full control (accepting and adding devices, creating an app, deleting an app, kick users out of the system) and should therefor be trusted by all the users in the network.

User

A user is a human who owns a display device like a smartphone, tablet or computer. The user only has to connect to the system once by entering the ip address of the server. they can always disconnect from a system and afterwards easily reconnect without entering the ip address again. The use of the applications should be very simple and therefor even a beginner is able to use the program. The minimum age for using the different applications of the system can be set to 12. At this age they should be old enough to not just play around with applications that, for example, can turn on and off lights. However, it is always possible for the administrator to block some applications for specific users.

3.6.2 Usability Requirements

Specifying some usability requirements will be of a great help to evaluate the program later on. It will also reflect the usability early in the development and provides concrete objects to be fulfilled. A division between functional requirements (what the system should do) and non-functional requirements (how the system should do it) is made. At the end of this section, some detailed information of certain requirements is given.

Functional Requirements

1. The product should only give access to authorised administrators.
2. The product can only be used by users/groups/devices the administrator granted access to.
3. The product should allow the administrator to manage things and display devices (add, delete, update).
4. The product should provide an overview of all things and display devices.
5. The product should allow the administrator to create, update and delete applications.
6. The product should allow the administrator to accept and deny devices that want to connect.
7. The product should provide a live overview of all connected display devices.
8. The product should provide a live overview of all applications in use.
9. The product should allow the administrator to kick display devices out of a session.

10. The product should provide an overview of all applications that can be used by the user.
11. The product should allow the users to disconnect from it.
12. The product should allow the users to distribute their current state to another device.
13. The product should allow the users to close an application.

Non-Functional Requirements

1. The product should give user feedback when performing actions.
2. The product should show a clear message in case of errors.
3. The product should show hints to the administrators to help them create applications.
4. The product should be easy to use, both for administrators and users.
5. It should not take longer than 1 minute to login to the administrator panel.
6. It should not take longer than 1 minute to connect a display device to the system.
7. It should not take longer than 2 minutes to connect a thing to the system.
8. It should not take longer than 2 clicks to access an application.
9. It should not take longer than 2 minutes to enter the basic properties of an application (name, devices, users).
10. It should take a training of maximum 10 minutes to know how to create a mashup.

Easy to Use Mashup Tool

When groups and displays are selected to participate in a certain application, the administrator can start developing the program. In order to make this mashup tool accessible for non experienced programmers, the tool should be as simple as possible with a minimum need for programming. This can be achieved by selecting and connecting different UI controls to perform the desired actions. For the more experienced users, some special attributes can be provided (e.g. a function UI component where the administrator can edit data the way he likes). Using flow charts is a popular way to develop mashups in as well the *first generation mashup tools* like Yahoo! Pipes [58] and Microsoft Popfly [40], as mashup tools for the Web of Things (Open Sen.se [56], glue.things [35]).

Possibility to Manage Displays, Devices and Sensors

The key is to work with multiple WoT devices and displays. The administrator should be able to add both of them. For displays (phones, smartwatches, laptops, ...), the users should be able to connect to the system by pressing a button on their device, in case they are not connected yet. Afterwards, the administrator can grant access to the users and assign them to one ore more groups if desired. Next, the list of applications is loaded on the screen of the users, who can see all applications they have access to. Applications that are not accessible for the user, should not be shown on the homescreen. Some extensions are possible, for example, the mashup tool can give the administrator the power to grant access for a certain period (an hour, for a week, forever, ...).

Next to display devices, other devices (like a Raspberry Pi which can control things, a lightbulb with wi-fi access, ...) should be able to be connected to the mashup tool. These could be devices in the local network, or somewhere else on the Internet. All of them have their own URL, which can be used by the administrator to add them to the system. At all time, the administrator should be able to delete a display or WoT device from the different lists.

Using Flowcharts to Create Mashups

With flowcharts, applications can easily be created by just dragging an dropping attributes to a canvas. Afterwards, it is possible to connect different elements to each other, which results in one big application. The administrator should be able to name the elements and give them a unique id. In this way, it becomes possible to 'catch' a certain element and perform actions with it (e.g. when a user clicks on 'ButtonA', do action 'Light Out'). Situations will exists where two attributes cannot be linked to each other, because it would result in an useless action. To help the administrator discover such actions, the mashup tool should give hints. This can be done by disabling certain attributes when an element on the canvas is selected, or to disable input and/or output ports of an element that is already on the canvas when another one is dragged towards it.

A challenge for the flowchart will be the creation of different user interfaces: how to align UI elements on a mobile phone? How to show them on a tv screen? One option would be to give the administrators as much freedom as possible and let them create a UI for each display. Another option is to define some rules in advance, and to let the program itself create the design of an application. Scores can be added to attributes to show them correctly to the user. As an example, the following situation can occur: assume there are two elements, a map and a button. The map should always be shown over the full width of the page, no matter which display device is in use. In this case, the administrator can enter the same score for each display. In case of the button, it will not always be stylish to show it over the full width of a page. Here, the administrator can specify to show the button over the full width of the page if the application is shown on a mobile phone, and to show a smaller button if the application is running on a laptop. An option should be offered to the administrator to edit these widths for each application separately, meaning that buttons can have different widths in different applications shown on an iPad.

As mentioned earlier, administrators should be able to design actions with the flowchart. If the administrator wants to create an application where a textbox and button are shown on a smartphone, and a map on a tv screen where it is possible to enter a city on the smartphone wich will be shown on the map on tv when the button is clicked, it should be very easy for the admin to implement by means of a special 'function' element. As administrators with no programming experience should be able to use this application, there is a need for a simple way to define these if-then structures. Next to actions, some filter elements should be available to filter sensor data. It is also important to notice to automatically update the look of certain applications if an action is performed. Assume users can turn on and off a light by clicking on a button on their phones. The text value of this button should be 'Turn light on' if the sensor data tells us that the light is off, and vice versa. If one users clicks this button, the text on this button should change not only for this user, but for all users that are using the app.

Node-RED[1] can be used as an inspiration for this flowchart. An extension of Node-RED, Node-RED-dashboard[2] shows some nice features to automatically create a dashboard application with different UI elements. In order to make the applications distributed, new elements should be added, with respect to characteristics given in [45]. How this can be achieved, is mentioned

[1]`https://www.nodered.org`

[2]`https://github.com/node-red/node-red-dashboard`

later in this chapter.

Managing Active Applications

When multiple users are using the same application, problems may arise. For example, multiple devices can join a session where photos of a mobile phone are sent to and shown on the television. When two or more users are continuously sending photos to the server, the user experience will be bad. The administrator then can kick one or more users out of the application if needed.

3.7 User Task Analysis

In this section, Concurrent Task Trees (CTTs) will be developed to identify and define the tasks of the users that need to be supported. CTTs are based on the hierarchical decomposition of a task and allow to express temporal relationships between activities. The graphical notation will give a better look how the program has to work and how tasks can be completed by the user.

3.7.1 Admin Tool

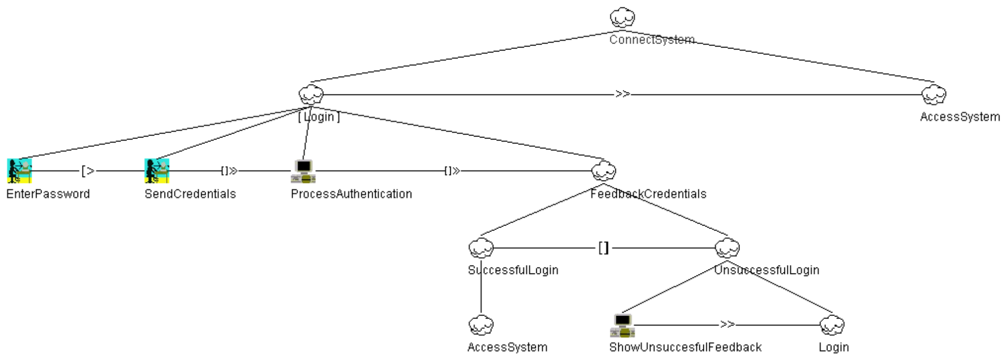


Figure 3.1: Administrator tries to login to the system

The admin tool allows an administrator to manage users, groups, applications and create mashups once the correct credentials are passed. Bellow, an overview of all possible actions is given. Notice that not all CTTs are shown here. Some of them are very similar to one that is explained earlier and are left out (e.g. creating and editing a group is very similar, only the CTT to create a group is given).

In Figure 3.1 the connection from an administrator to the system is depicted. If the administrator is not logged in yet, a password has to be entered. This will be ‘admin’ the first time someone tries to connect to the system. Later on this password can be changed by the administrator. A negative feedback is shown when the password is incorrect.

Manage User Requests

Once the administrator is logged in, the system can be used. The actions that can be performed are described in Figure A.2. In Figure A.1, the situation is depicted what will happen if a new user tries to connect to the system. All current tasks will be stopped until the administrator has made a decision. The admin can accept the user by entering a username, selecting a type and connect the user to some groups and applications. Validation errors are shown if needed. The admin can also choose to deny access to the user. The system will send an appropriate message to the user and will continue with the task that was paused.

Using the System

As long as no new user tries to connect, the administrator can use the system in the preferred way. Choices can be made to see the homepage, see users (the different display devices), groups, WoT devices or applications. It is also possible to change the password or logout. The CTT can be found in the Appendix, Figure A.2.

Homescreen

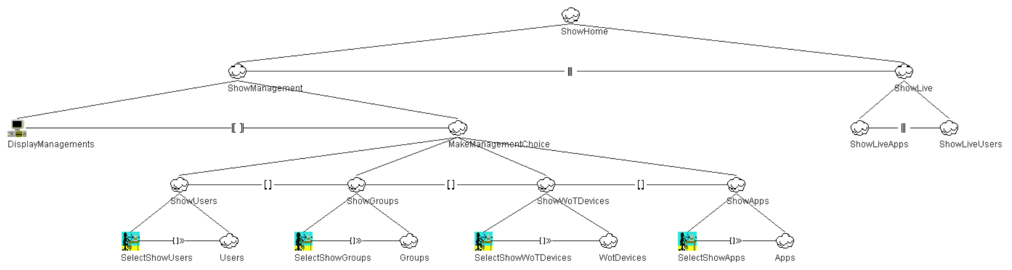


Figure 3.2: Possibility to manage different kinds of data and see live applications and users

The CTT in Figure 3.2 gives an overview of all the possibilities when connecting to the administration tool. On the homescreen, the admin can choose to manage users, groups, devices or applications. When clicked on a link, the system will redirect to the correct page. Also shown on the homescreen are live applications and online users.

Homescreen - Stop Live Apps

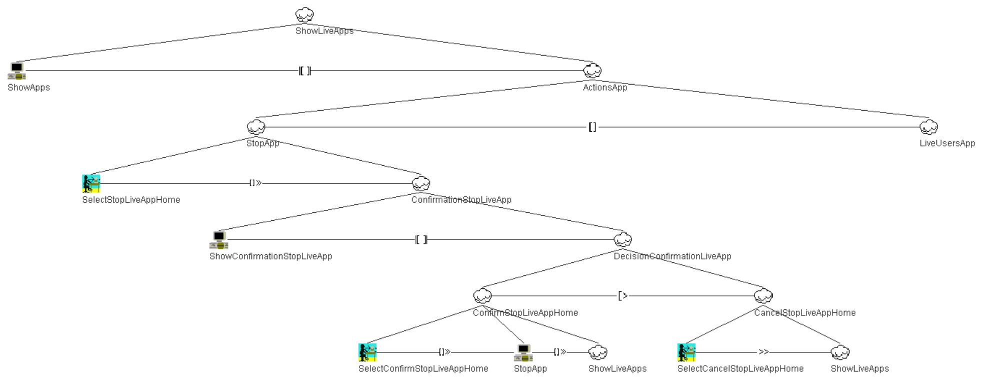


Figure 3.3: Stopping a live application

From the list of live applications, one app can be chosen and stopped if desired. The administrator is asked for a confirmation once the stop button is clicked. The CTT in Figure 3.3 shows this process.

Homescreen - User Using a Live App

The administrator can view all the users that are currently using the live application, as depicted in Figure A.3. If a user has to be removed from a live application, this is possible: a confirmation is asked to only kick the user from the session, or also block the user from using the app again.

Homescreen - Live Users

Figure A.4 shows also an overview of all live users is provided, together with the application they are using (if applicable). Users can be stopped from using the application, or blocked entirely from the system after confirmation.

Users

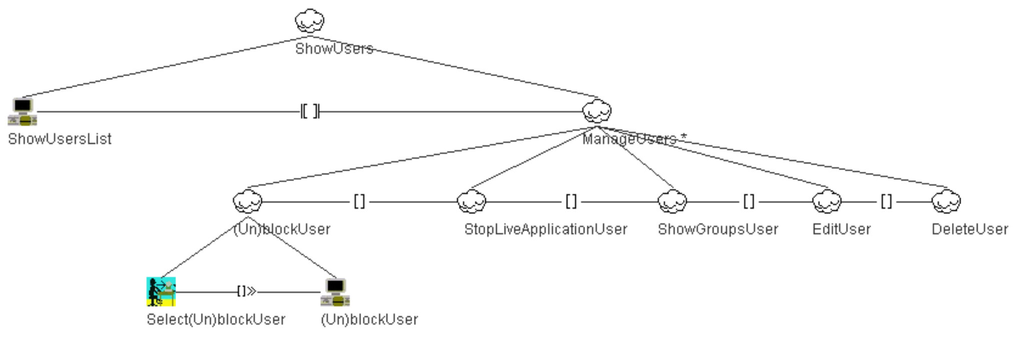


Figure 3.4: A list of all users is provided. Actions to update, block or remove users are available

The administrator can ask for a list of all users in the system. It is shown whether a user is online or not. If the user is online and using an application, this application can be stopped. This will happen in the same way as in the previous subsection. Users can also be blocked, edited or removed. A list of groups the user is connected to can be shown. All of this is displayed in Figure 3.4.

Users - Show Groups

Users can be connected to multiple groups. An overview of all connected groups can be requested. The administrator has the possibility to add a new group to the user (redirecting to the 'Edit' page), or remove a group from the list. Confirmation in case of removal is asked. The CTT is given in Figure A.5.

Users - Edit

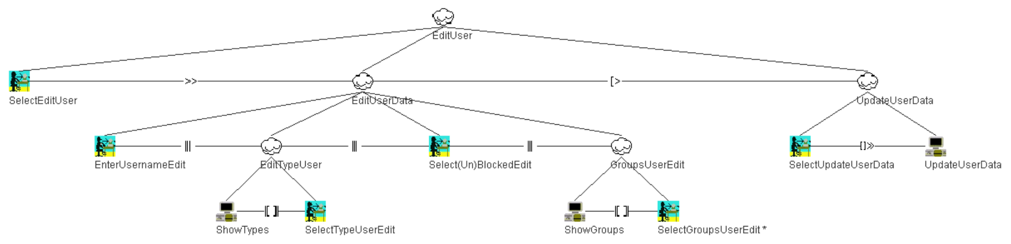


Figure 3.5: Updating a user

Figure 3.5 shows a user can be edited by changing the name, select another type, (un)block the user, or add/remove some groups to/from the user. An update button sends all the new data to the server.

Users - Delete

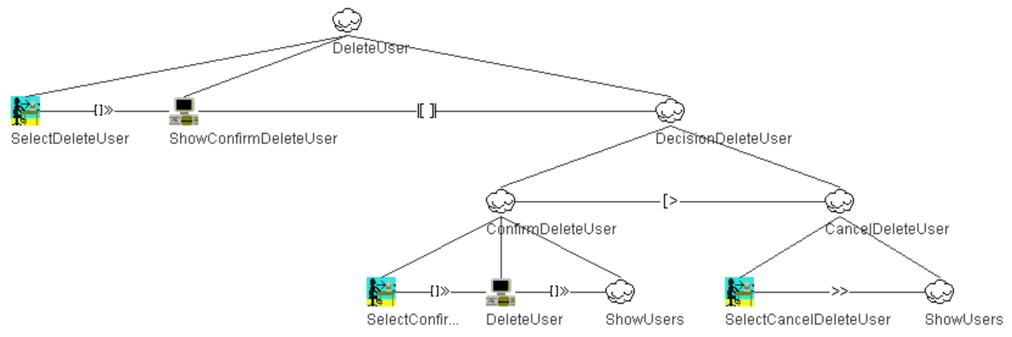


Figure 3.6: Deleting a user

If the administrator choses to delete a user, a confirmation screen will pop up. The admin then can accept or cancel the deletion of the user. The process is shown in Figure 3.6.

Groups

A list of groups will be shown together with some actions that can be performed: showing users connected to a group, edit a group, remove a group and create a new one (see Figure 3.7). Since showing users of a group is similar to showing groups of a user, this CTT is skipped. In Figure A.5 a similar CTT can be investigated. Removing a group is similar to removing a user, so this CTT is also not provided. See Figure 3.6 to get an idea how it works. Next, the creation of a new group is given in Figure 3.8. As updating a group is done in a similar way, the CTT of updating a group is skipped.

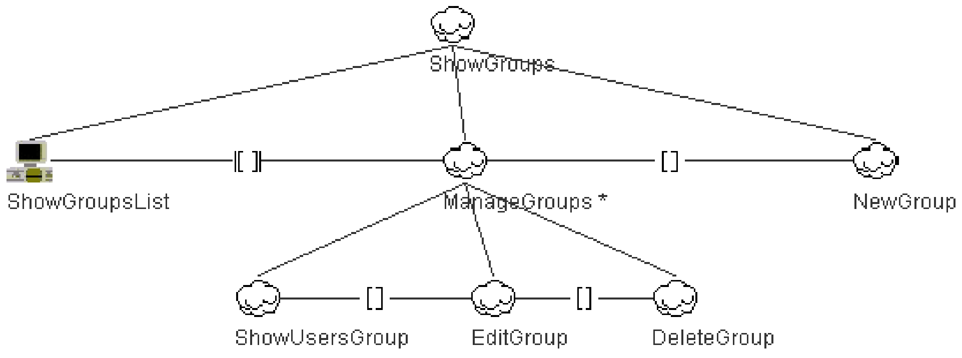


Figure 3.7: Showing groups

Groups - New

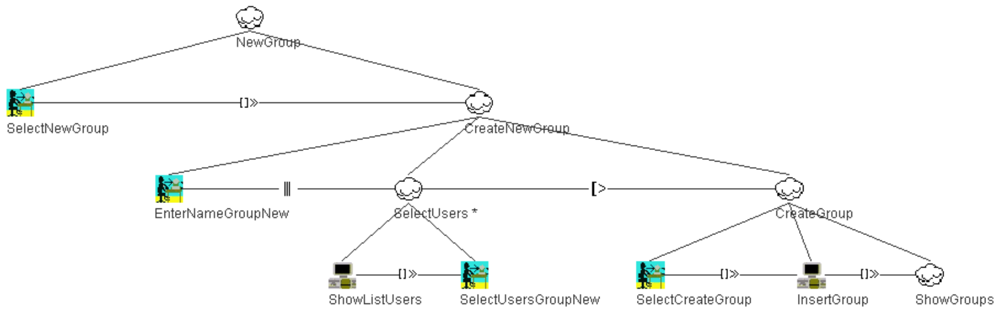


Figure 3.8: Creation of a new group

To create a new group, the administrator has to pass a name and select some users who will be connected to the group. Once this is done, the group can be created. As mentioned earlier, updating a group is very similar and will not be discussed in detail.

WoT Devices

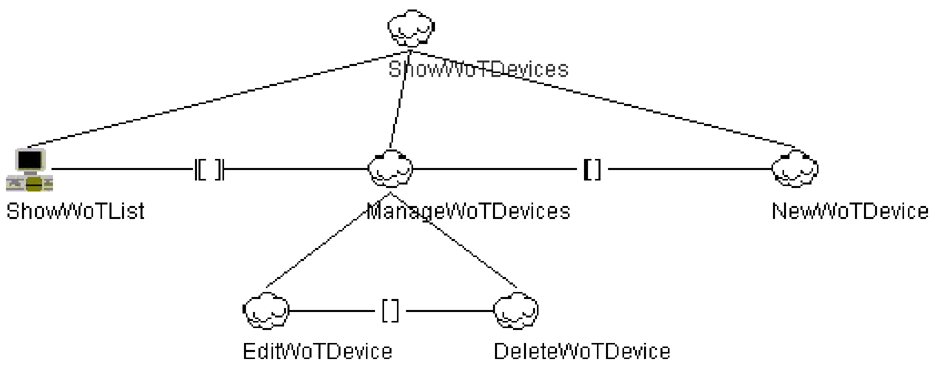


Figure 3.9: Overview of WoT devices

Selecting to show WoT devices will give the administrator an overview of all the things that are already inserted, as shown in Figure 3.9. Devices can be deleted and updated, and a list of all the getters and setters of a certain device can be asked. The CTTs to update and delete a WoT device will not be shown. Next, the CTT to create a WoT device and assign getters and setters to it are given. Choosing to view all the getters and setters of a certain WoT device by selecting it in the list, will result in a redirect to the getters and setters page of a WoT device, which is shown in Figure 3.11.

WoT Devices - New

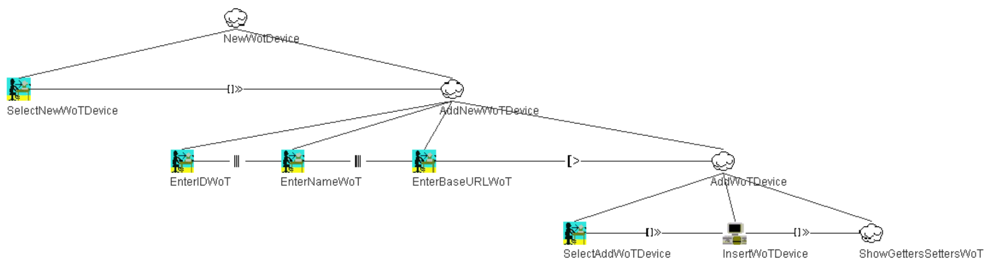


Figure 3.10: Adding a new WoT device to the system

Depicted in Figure 3.10 is what happens when adding a new WoT device. The administrator has to pass an ID, a name and a base url. This base url will be the url to control the device and get data from it. When selecting the action to add the device, the administrator is redirected to an overview of all the getters and setters of this device (which will be empty in this stage).

WoT Devices - Getters and Setters

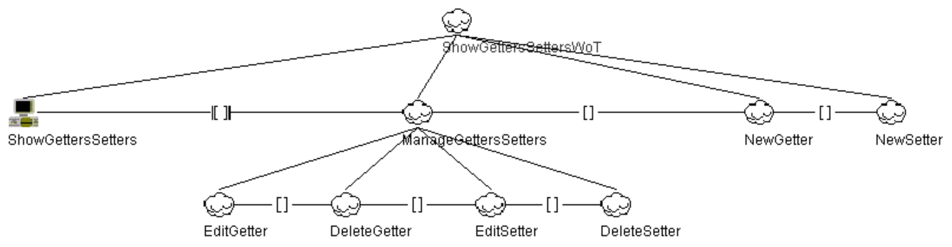


Figure 3.11: Overview of all the different getters and setters of a WoT device

A list of all different getters and setters for one WoT device will be shown. An administrator has the possibility to delete or update existing getters and setters), or to add new ones (see Figure 3.12 to add a getter and Figure 3.13 to add a setter).

WoT Devices - New Getter

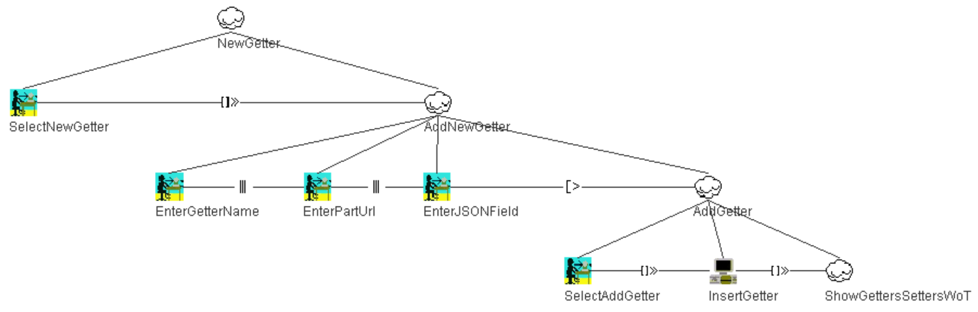


Figure 3.12: Adding a new getter

To add a new getter, several fields have to be filled in. The admin should pass a unique name, a part of a url and a JSON response field. The part of the url should be the part that comes after the previously defined base url. If both urls are correctly inserted by the admin, the combination of them should result in getting a JSON object. To get the correct value of this JSON object, the passed JSON field is used. Updating a getter is similar and this CTT will not be given.

WoT Devices - New Setter

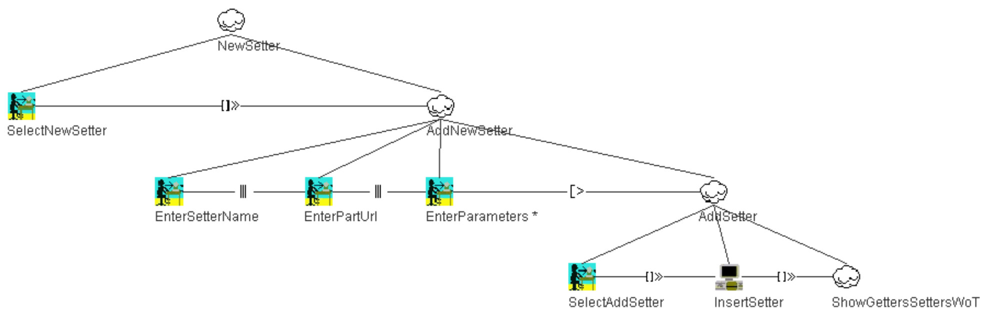


Figure 3.13: Adding a new setter

Adding a new setter is very similar to adding a new getter: a unique name has to be passed, together with a part of a url. The only difference is that several json fields can be passed. These will act as parameters when using this setter. Again, updating a setter is very similar and will not be discussed in detail.

Apps

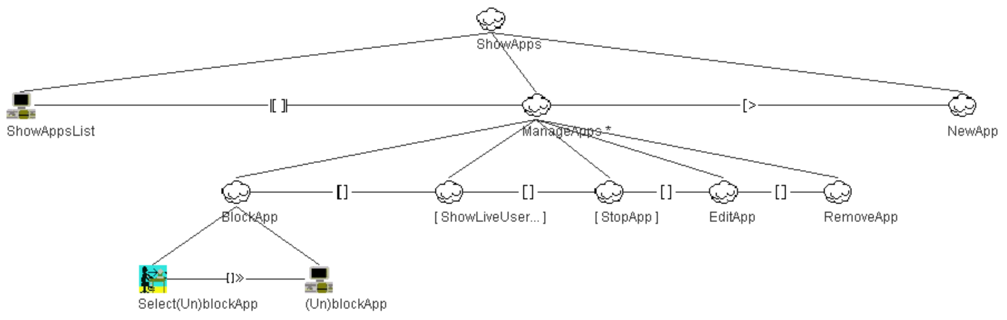


Figure 3.14: Overview of applications

An overview of all applications is shown when selecting the Apps page. An admin can block an application, edit or remove one, or create a whole new one. If an application is live, the app can be stopped or a list of active users can be asked. Active users can be kicked out of the session, or blocked entirely from using the app again, as displayed in Figure 3.14. These actions are similar as in Figure 3.3 and Figure A.3 and will not be discussed again.

Apps - New

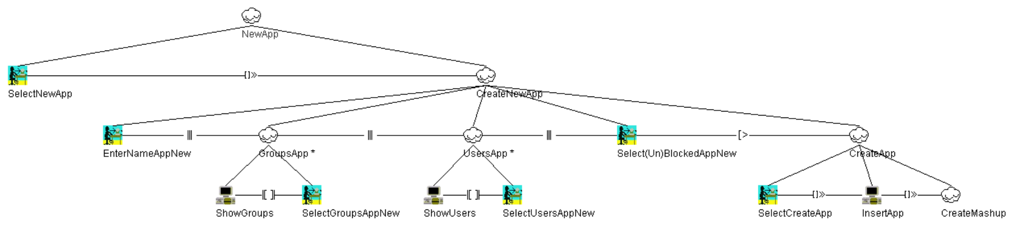


Figure 3.15: Creating a new app

Creating a new application (Figure 3.15) requires a name, a blocked status and some users connected to the app. A list of groups is provided: when a group gets selected, all the connected users get selected. These users can be deselected individually afterwards. When the admin selects ‘Create App’, the app gets created and the admin gets redirected to the mashup creation page. Editing an app is very similar and thus will not be shown here. The only difference is that updating an app does not result in a redirect to the mashup. Instead, a link to the mashup is provided on the update screen.

Apps - Mashup

Figure A.6 displays the creation of a mashup. All available elements that can be dropped to a canvas are shown. When dropping an element to the canvas, the admnistrator has to provide some information: this is different for each element, the only thing they have in common is an ID. The admin can confirm the element and afterwards the element will be dropped on the canvas (if validation checks are ok). The admin can also cancel the operation. Elements on the canvas can be moved and edited (which open the same window as when inserting a new element, but with data filled in). When the user is satisfied with the mashup or just wants to save the work, the ‘Deploy’ option can be used.

3.7.2 Hasys

'Hasys' stands for HomeAutomationSystem and is the app users can work with to see the different applications made by the administrator. Below, an overview how the users should connect to the system, how they can use applications, how they can distribute and stop applications and how they can logout from the system is given.

Connecting to Hasys

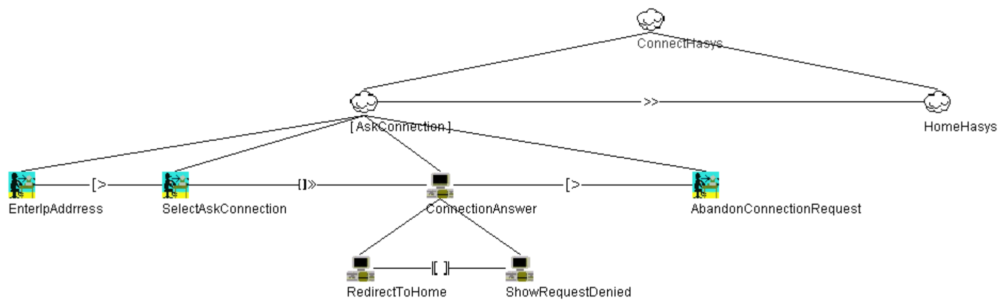


Figure 3.16: Sending a connection request to the system

When a user wants to connect to Hasys, a check is made if the users is not already connected to the system. If so, the user is redirected immediately to Home. If not, the user has to enter the ip address of the server Hasys is running on. The administrator will receive this request and decides if the user may connect to the system or not. While waiting for an answer, the user can always abandon the connection request. All these tasks are depicted in Figure 3.16.

Homescreen of Hasys

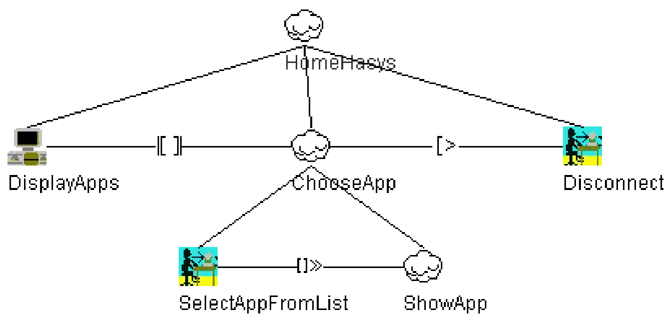


Figure 3.17: Displaying the home screen of Hasys

Once the user is allowed to use the system, a list of all possible applications is shown from which the user can choose (see Figure 3.17). After choosing, the correct app will be displayed to the user. A disconnect button is available to logout from the system.

Using an Application

After selecting the preferred application, Hasys will generate the correct application. When using the app the user can perform some generated actions, or can choose to distribute the application to another device. A close button is present to cancel the action to distribute. When the user is done using the application, he can click on the right button for it. The CTT of these tasks can be found in Figure A.7.

3.8 User Object Modeling

To identify and understand the objects users will manipulate during their tasks, an ORM diagram is developed. In this ORM diagram different entities exist and the relationships between them are shown. The complete ORM diagram can be found in the Appendix, Figure A.8. In this section, the ORM will be described in detail. To make things readable, the ORM diagram is split up in different parts.

3.8.1 Users and Groups

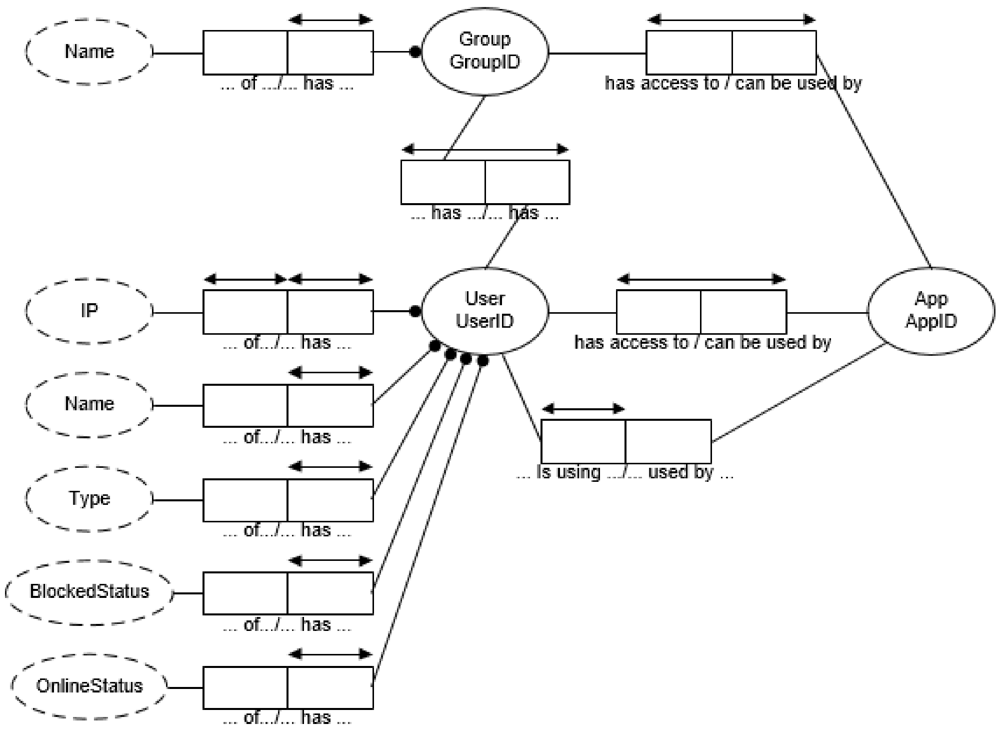


Figure 3.18: User and Group entities

A group only has an ID and a name as value. It can be connected to multiple users, and can be selected when creating an application. Users on their turn can be connected to multiple groups and have access to multiple applications. Each user is actually a display device that is connected to the system. Therefore, an IP address for each user will be saved. Furthermore, a name, block status, online status and type is stored. The type will be the device type, which can be a smartwatch, smartphone, tablet or laptop. All of these are mandatory.

3.8.2 Applications

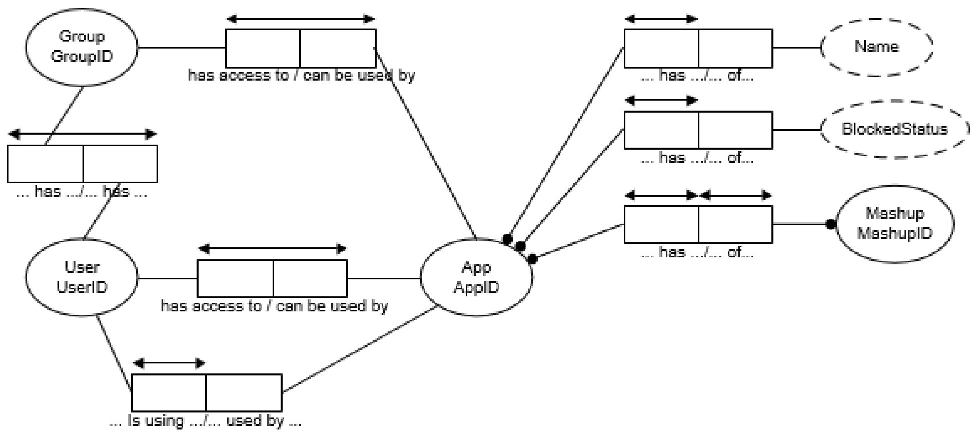


Figure 3.19: Application entity

As mentioned before, an application can be used by different groups and users. Each application also holds a list of all the live users that are currently using the app. None of these are mandatory. Next, a name and block status is given for each app. To create an app, a mashup should be made. Therefore a mashup ID will be connected to the app. This will be a one-to-one relationship.

3.8.3 Mashups

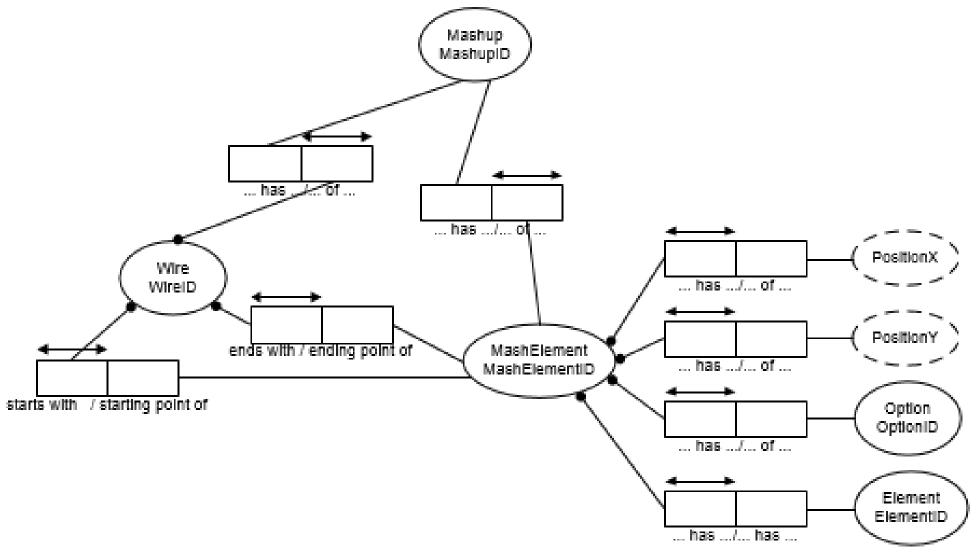


Figure 3.20: Mashup entity, consisting of MashElements and Wires

A mashup exists of several elements that can be connected to each other. One element can be connected to multiple other elements, but this is not mandatory. The connection is stored in a Wire entity, which expects a MashElement as starting point and another one as ending point. These points are mandatory: a wire cannot exists without these points. A MashElement is an element, together with some special options. These options depend on the type of element and are not described in detail in the ORM diagram. An example can be the options `value`, `placeholder` and `ID` for a textbox element. For each MashElement the PositionX and PositionY are stored to draw it correctly on the canvas.

3.8.4 Elements

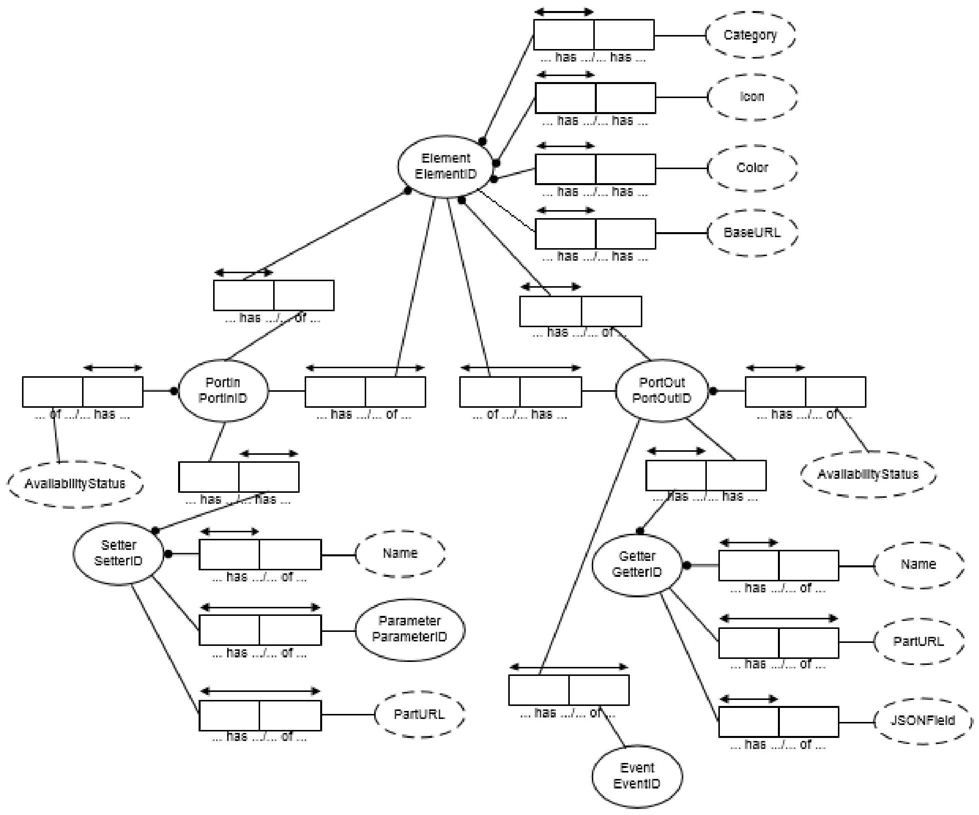


Figure 3.21: Mashup entity, consisting of MashElements and Wires

All elements that can be used in the mashup tool will be stored. Each element has an ID, a name, an icon, a category, a colour and the number of times the element can be dragged to the canvas while creating a mashup (for displays, this is limited to one time, ui elements can be dragged multiple times). Possible categories are ‘display’, ‘ui’, ‘function’, “sensor” and “wot”. All these fields are mandatory. An optional field for some elements will be the BaseURL. For WoT devices, this field can be used to store the url to connect with a device.

Also the input and output port of each element is stored. For both type of ports the availability status and elements they can connect to are described. In case of a UI element, sensor or WoT device, it is possible to assign setters to the PortIn entity. A setter exists of a mandatory name, some parameters (optional) and in some cases a PartURL. This PartURL will be used when the element is a WoT device and stores a part of a url (the one that comes after the base url) to access the device. For the PortOut, some getters can be added. A getter has a name, and in the WoT case a PartURL and a JSONField. This JSONfield will store a field from the JSON response that will be requested. Next to setters, an event can be connected to the PortOut if needed. For example, the event 'onclick' can be added to the PortOut of a ui button element.

All these getters, setters and events will play an important role when defining functions inside a mashup. More detailed information can be found in the 'Implementation' chapter.

3.9 Style Guide

A good style guide will result in a consistent UI style and increases productivity because users will make less errors and will learn faster how to use the system correctly. This thesis will use Bootstrap v4.1 [3] to create web pages, meaning the style will be based on the general Bootstrap style guidelines.

3.9.1 Standards for Window Interaction

- A navigation bar on top will allow the user to navigate between different windows. This navigation bar will always be fully expanded and be fixed at the top of the page.

- Some buttons are provided in the navigation bar to distribute or stop an application.

- Clicking on a link will result in opening a dialog box, or going to another page.

- On the homescreen, shortcuts are provided to go another page.

- A dialog box can be closed by pressing a cancel button, or to confirm the action that is being asked for in the box.

- Clicking on an item in a list box, will result in a redirect to the correct application.

[3] `https://getbootstrap.com/docs/4.1/components/`

3.9.2 Standard Window Layout

The main window has a navigation bar on top which will always be expanded and fixed to the top of the page. This bar has links to other pages, and in case of the user some buttons to perform particular actions like distributing an applications or stop using one.

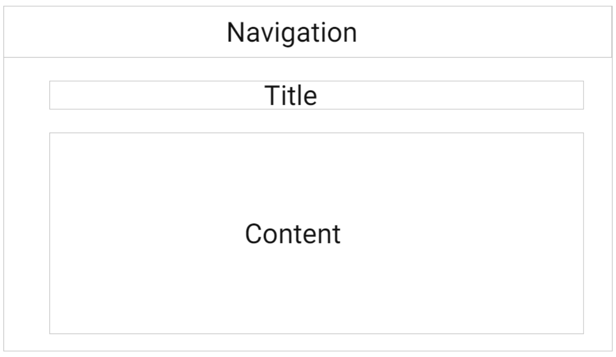


Figure 3.22: Standard Window Layout

In most cases, the title of the page is present bellow this navigation bar, following by the actual content of the page as shown in Figure 3.22. This content area can have a table, a list box or some buttons, depending on which page is displayed.

For the creation of an actual mashup, the window layout is a bit different as depicted in Figure 3.23. The navigation bar is still present, but the rest of the page is constructed differently. On the left side, a list of elements will be given which can be dragged to the canvas on the right. On top of this canvas, an action to deploy the constructed mashup is provided.

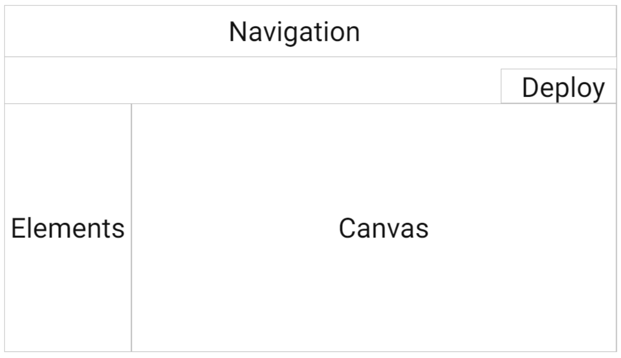
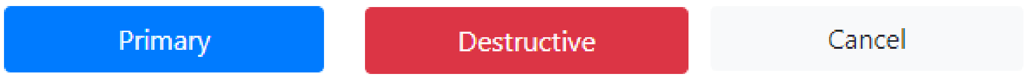


Figure 3.23: Mashup Layout

3.9.3 Standards for Menus and Push Buttons

The Bootstrap library provides a predefined set of buttons. The primary button (blue colour) will be used to perform actions that result in inserting new data or alter existing data. To cancel operations, the light button (light grey colour) is being used. Buttons or icons that link to a destructive action like removing an application are using the danger colour (red). Each time when performing a destructive action a confirmation is being asked. To confirm the action, one has to select a primary button to definitively perform the destructive action, or select the light button to cancel the operation. An overview of these buttons is give in Figure 3.24.



(a) Primary button (b) Destructive button (c) Cancel button

Figure 3.24: Different types of buttons

The Bootstrap v4.1 standards for buttons [4] and navigation menus [5] can be found on the site.

3.9.4 Standards for Use of Keyboard Keys

Since both applications will be created using web technologies, the standard keyboard keys that can be used using a browser will be available for the users and administrator. For example, when a number has to be inserted in a number input field, the arrow up and arrow down keys can be used to alter the number in the input field. When constructing a mashup, the delete key can be used to remove the currently selected wire or element.

3.9.5 Standard Use of Colour, Type and Fonts

As mentioned earlier, the primary colour of Bootstrap (blue) will be used to perform actions that result in inserting or altering data. The light colour (grey) will cancel operations. Destructive actions will have a danger colour (red) assigned. This danger colour will also be used to show validation errors. The success colour (green) will be used to show successful messages. The other possible colours like info (light blue) and warning (yellow) will be used

[4]https://getbootstrap.com/docs/4.1/components/buttons
[5]https://getbootstrap.com/docs/4.1/components/navbar

Name	Colour	HEX code	Actual colour
Primary	Blue	#007BFF	example
Secondary	Gray	#6C757D	example
Success	Green	#28A745	example
Danger	Red	#DC3545	example
Warning	Yellow	#FFC107	example
Info	Light blue	#17A2B8	example
Light	Light grey	#F8F9FA	example
Dark	Black	#343A40	example

Table 3.1: Overview of predefined Bootstrap colours

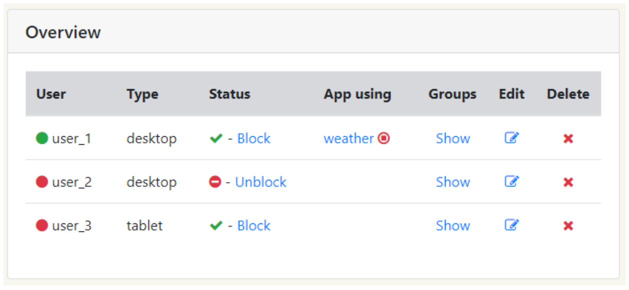
in various ways. In Table 3.1 all the colours provided by Bootstrap, together with their HEX code are given.

A complete overview of the Bootstrap Text/Typography can be found on the site of w3schools [6]. The default font-size of Bootstrap is 16px, with a line-height of 1.5. The default font-family is set to 'Helvetica Neue', Helvetica, Arial, sans-serif. Bootstrap provides 6 different headings, going from <h1> to <h6>. <h1> has a font-size of 40px, <h2> 32px, from hereon the amount of px drops by 4px for each following heading. Each heading has a bolder font-weight to be distinguishable from normal texts.

3.9.6 Standards for Use of Tables

The table header of a table will have a grey colour and headings will be in a bold style to make a separation from the other rows containing data. The table body holding all the data will have a white background. At the bottom of each row, a thin light grey line is added to clearly see the difference between two rows. Data cells that always will have the same data within it (like 'Edit', 'Delete' and 'Show'), will be assigned a fixed width that is just enough to save space for other data cells. Icons will be used where possible to save even more space (e.g. a red remove sign to fill a delete cell). An example of such a table can be found in Figure 3.25.

[6]`https://www.w3schools.com/bootstrap4/bootstrap_typography.asp`



Overview

User	Type	Status	App using	Groups	Edit	Delete
● user_1	desktop	✓ - Block	weather ◉	Show	✎	✖
● user_2	desktop	⊖ - Unblock		Show	✎	✖
● user_3	tablet	✓ - Block		Show	✎	✖

Figure 3.25: Table containing all the users. Icons and/or small cell widths are used on the 3 last cells to save some space for the other cells

3.9.7 Standards for Use of Data Types

- For booleans a single checkbox will be used.
- For the input of number, a number field will be present.
- When making a choice out of less than four options, a radio button group will be used.
- When the number of possible choices is bigger than three, a dropdown list is chosen.
- For choices accompanied by a picture, radio buttons are used (e.g. a list of possible icons to assign to a WoT device).
- For all other possible types of input, a standard textbox is being used.

3.10 Prototyping

The last step in the design process is to create the user interface, based on the models and the style guide. First some drawings on paper were made and after an evaluation online mockups were made on MockingBot [7]. Several of these (uncoloured) mockups can be found in the Appendix. Based on these mockups, the final application was created.

[7] https://www.mockingbot.com/

4 Implementation

Following the User Interface Design process, the final step is to develop the application. In this chapter, all the technical details about the implementation of the mashup tool will be given. This mashup tool should be able to solve the shortcoming of current mashup tools for Distributed User Interfaces (DUIs) and for the Internet of Things (IoT), namely integrating both DUIs and IoT into one mashup tool. To start with, an overview of the used technologies is given, together with some example code to receive and send data from/to a server. Afterwards, a section about creating the mashup itself using jQuery and svg elements is discussed. To end with, it is shown how an interpreter on the user side will draw all applications correctly and perform defined functions in the right way.

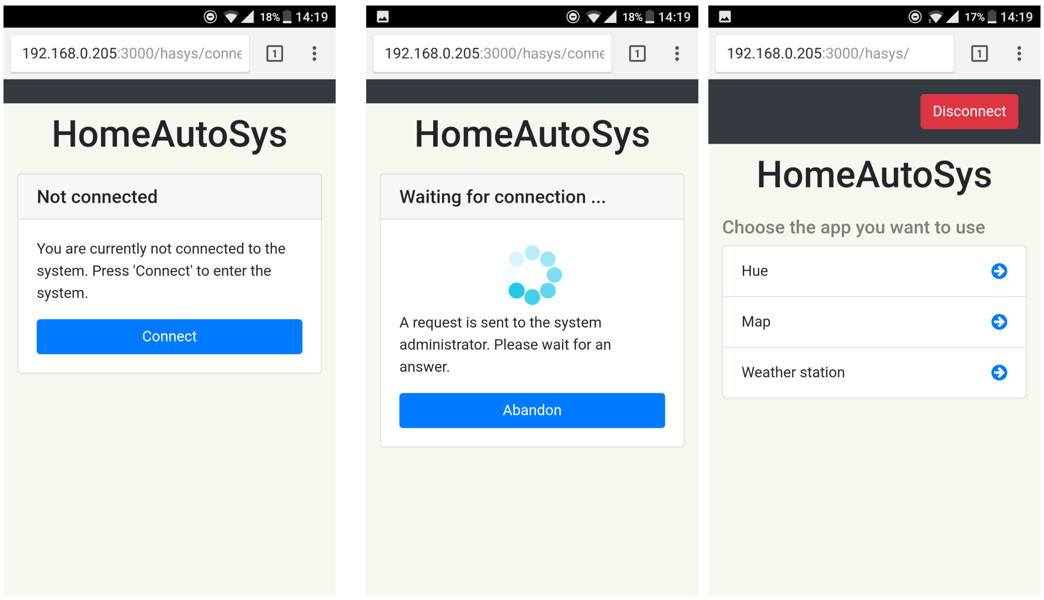
4.1 Choosing for the Web

The first question was *how* this application should be created. On the user side, multiple options exist: one could create separate Android and iOS applications, or could choose to use a framework like Ionic[1] to nicely create applications for both Android and iOS at the same time. A third option was to just use the browser on mobile devices and create web applications. As the

[1] `https://ionicframework.com/`

third option was most interesting for this thesis, this one was chosen. Native web technologies could be used and users should only have a web browser on their device to make the application work. The same code will work on as well a tablet, a smartphone as a computer. On the admin side, also multiple options were possible like programming in Java or C# and making a connection to a server. However, for this part the choice for the Web was also made. This way the same programming languages are used for all the different parts and one should only know web languages like HTML, CSS and JavaScript to extend the system.

4.1.1 Designing Applications



18% 14:19
192.168.0.205:3000/hasys/conne
HomeAutoSys
Not connected
You are currently not connected to the system. Press 'Connect' to enter the system.
Connect

18% 14:19
192.168.0.205:3000/hasys/conne
HomeAutoSys
Waiting for connection ...
A request is sent to the system administrator. Please wait for an answer.
Abandon

17% 14:19
192.168.0.205:3000/hasys/
Disconnect
HomeAutoSys
Choose the app you want to use
Hue
Map
Weather station

(a) Connect to system (b) Waiting for approval (c) All accessible apps

Figure 4.1: The user goes to the correct url. When the user is not connected to the system yet, he has to press the 'Connect' button (a). After sending the request, the user has to wait for the administrator to accept it. It is always possible for the user to abandon the request (b). When accepted by the administrator, a list of all possible applications for this particular user is shown

Before setting up a server and letting things communicate with each other, the design of both the user and administration applications were done. To do this, basic web technologies like HTML and CSS were used. To get some help, the popular toolkit Bootstrap v4.0[2] is used. Bootstrap offers front-end components which quickly and easily can be used to create web applications.

On the user side, the design of the application got some minor changes compared to the final mockups. Since the user should surf to the correct url in order to login to the system, there is no need to provide an extra textbox for the user to enter the ip address of the system. A simple button with the value 'Connect' is enough to connect to the right system. In Figure 4.1, the 3 main screens of the user application are shown: the connection screen, the screen while waiting for the administrator to accept the connection, and the home screen where all the available applications for the particular user are listed and can be chosen to use. On the admin side, no great changes took place. Further in this chapter, the most tricky part (the creation of the mashup tool itself) is discussed in detail.

4.1.2 RESTFUL Web Service

```
var http = require('http');
var express = require('express');

var app = express();
var server = http.createServer(app);

server.listen(3000, function(){
 console.log('Server␣started␣on␣port␣3000');
});
```

Listing 4.1: Setting up a simple Express.js server

In order to provide some interaction in both applications, a server should be set up which can share data over the different systems. As said before, a RESTFUL web service will be created in order to do so. In this project, Express.js[3] is used to develop an API to share data with others. Express.js is a lightweight Node.js[4] web application that provides all the necessary features to setup a RESTFUL web service. In addition to all the great features, it is also very easy to install. One should first install Node.js and afterwards use the package manager (npm) to import the Express.js package. Once this is done, only a few lines of code are needed to define the server (see Listing 4.1). First, an index file for the server should be created. In this case, this index file will be called `server.js`. In this file, Express has to be required and invoked afterwards. Likewise, http should be required and could then be used to create a http server by using the invoked express variable. The user

[2]`https://getbootstrap.com/`
[3]`https://expressjs.com`
[4]`https://nodejs.org`

now only has to specify the port the server is running on, and a first test can be performed. The server can be started by running the command `node server.js` in a console.

Storing Data

```
{
 "users" : {
  "5pmcs1th47d28qh" : {
   "ip" : "192.168.0.240",
   "name" : "user_1",
   "type" : "desktop",
   "blocked" : false ,
   "online" : false ,
   "activeApp" : false ,
   "groups" : [
   {
    "groupId" : "1vwp2lhzoic8364",
    "name" : "group_1"
   }]
  },
  "mjyo5p03mdbsulb" : {
   "ip" : "192.168.0.102",
   "name" : "user_2",
   "type" : "desktop",
   ...
```

Listing 4.2: A part of the users.json file

To store data, the choice was made to just use simple JSON files. In the future, other solutions like storing data in database like MongoDB[5] or CouchDB[6] are possible. The following JSON files exist:

- `users.json`: stores all the information about the different users that are using the system.
- `groups.json`: a collection of all the available groups.
- `applications.json`: all the applications offered by the system. It contains info about the live users, which users are allowed to use the application, details about the mashup that creates the app, ...
- `elements.json`: contains all the elements that can be used when drawing on the canvas. Stores the color of elements, the icon, and all their options.

These will all be stored in the folder called 'data'. For each of these files a separate resource model is made. This resource model will import the JSON

[5]`https://www.mongodb.com/`
[6]`http://couchdb.apache.org/`

file and export it as a node module that can be used in the application. Before diving in on how to access this data, an example of such a JSON file is given. Below, a part of the users file is depicted.

Listing 4.2 shows two users that are already connected to the system. They both have a unique id that is generated on creation time. Further, each user has a name, a device type and the ip address of this device. The type property will play an important role when the user enters an application. Based on the type, the UI elements can be aligned in different way, depending on how the administrator specified it in the mashup tool. Also some information about the groups the user is connected to is stored (id and name), together with some real time information thanks to different status variables (blocked or not, online or not, which app is the user currently using).

Receiving and Sending Data

By setting up a REST web service, data can be retrieved by entering the right url and sending the right HTTP method (GET). To alter data, http methods POST, PUT and DELETE can be used in combination with a url. Using Express.js, it is not that difficult to create a complete API that responds to client requests. One could easily program all the possible endpoints in the server.js file, which is a popular method for doing so. But when creating a bigger application, things would become a bit unclear due to the large amount of code. To solve this, Express.js offers Routing[7]. The goal is to create a couple of routing files that handle client requests. In this project, one routing file per JSON file is used. This means there are 4 routing files available: `applications_routes.js`, `elements_routes.js`, `groups_routes.js` and `users_routes.js`. In Listing 4.3 a piece of code is given where a new router gets defined with a GET method.

```
var express = require('express');
var router = express.Router();
var data = require('./../data/users_model'); // import user data model
app.get('/:id', function (req, res){
 var id = req.params.id; // get the id parameter out of the url
 res.send(data.users[id]); // send back all information about the user
}
```

Listing 4.3: Routes: GET method with a parameter

In each file different methods can be defined. Each method is derived from a HTTP method and attached to an instance of the express class. The following listing shows how a GET request for a user with a specific id is

[7]`https://expressjs.com/en/guide/routing.html`

done. In this method, the developer can extract the parameters from the url and perform actions with it. To give the client access to this url, the route has to be imported in the `server.js` file, as shown in Listing 4.4. Once this is done, a client can easily send a request to localhost:3000/users/abc and receive all information about user *abc* in JSON format.

```
// Importing express and http
...
var users_routes = require('./routes/users_routes'); // import user routes

// invoke express, set up the server
..
app.use('/users', users_routes); // define route
}
```

Listing 4.4: Server.js: importing routes

Setting up routes for POST, PUT or DELETE is very similar. One just has to replace the *get* method by the desired method. To be able to work with POST parameters, one should first import the `body-parser` package in node.js. This package can then be used to extract POST parameters by the command `var name = req.body.name`.

4.1.3 Calling the API

Now that a web service is up and running, the clients can send requests to it. An example can be to send a request for getting all the users, and show these users afterwards in a HTML table. To do so, jQuery[8] is used. jQuery is a JavaScript library that makes it easier to code with JavaScript due to its easy-to-use API. Ajax, a part of the jQuery library, can be used to send requests to a server and receive data. This data can then be used to fill a table for example. Next, a small piece of code is given in Listing 4.5 where when the document is loaded, the function `get_groups()` is called. Inside this function, a call to `/groups` is made. The web service will handle this request and send back a list of all the groups in JSON format. The `success()` function of Ajax is triggered and will append all the users to the desired table. In this case, there is a table with a table body with the id `groups` somewhere on the HTML page. It is this table body that will be filled with all the groups.

```
$(document).ready(function(){
 get_groups();
});

function get_groups(){
 $.ajax({
```

[8]`https://jquery.com/`

```
   url: '/groups',
   method: "GET",
   crossDomain: true,
   dataType: 'json',
   success: function (data) {
    $.each(data.groups, function (index, item) {
     var group = '<tr␣id="'+index+'">'
       + '<td>' + item.name + '</td>'
       //
       // table cells with buttons to edit, remove groups.
       //
      + '</tr>';
     $('#groups').append(group);
     });
    }
   });
  }
}
```

Listing 4.5: Ajax request in jQuery to fill a table with all available groups

4.1.4 Real-time Updates

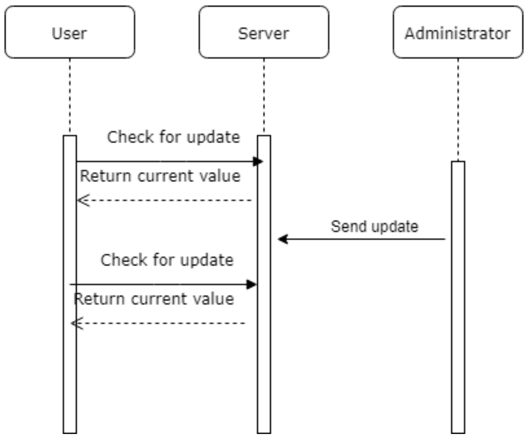
The developed system will be a real-time web application, meaning that when an important action occurs, the right clients should be informed as soon as possible. An example can be a user that gets blocked by the administrator: when this happens, the user should immediately be kicked out of the system. Another example is the communication between different devices. In a distributed user interface it is important that all devices are in the same state at each moment.

One option to cope with problem can be polling for new data each x seconds in JavaScript. In the case of the blocked user, the client can send each five seconds a request to the server by firing the function `am_i_blocked()`. This methods has it drawbacks: one problem is that an update will not take place immediately. If the JavaScript function just started his new round of, let us say five seconds, the user has to wait five seconds before the update finds place. In a distributed interface environment, this will make the program look slow and therefor have a negative impact on the user experience. Another problem is that useless requests will be made each five seconds when no updates are made.

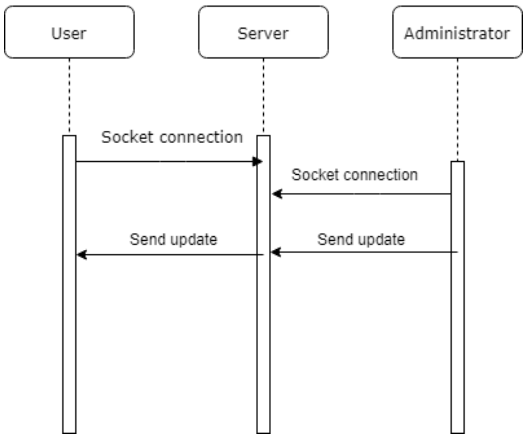
Luckily there is a solution to overcome this problem: WebSockets[9]. WebSockets are part of the HTML5 specification. With sockets, a permanent connection between clients and the server is made. This connection will always be open and the server can easily send messages to the client and vice versa. This means that when an administrator blocks a user, this user will immediately

[9] `http://www.websocket.org`

be kicked out of a session. When using a distributed user interface, an action on device A will send a message to the server, who will distribute the message to all or some devices in the system. Within a second all devices are up to date and in the correct state. In the pictures (Figure 4.2) below, the difference between polling with JavaScript and using WebSockets is given.



(a) Polling



(b) WebSockets

Figure 4.2: With polling (a) the users checks each interval for updates. When the admin sends an update, the user will see it in the next interval. With WebSockets (b), both the admin and user make a socket connection to the server. When the admin sends a message to the server, the server broadcasts it to the connected users

Introducing sockets in this project is easy when working with Node.js. On the Express.js server, the `socket.io`[10] library needs to be required in the `server.js`. Further on the possible messages need to be defined. On the client side, the `socket.io` JavaScript library[11] needs to be imported. This can simply be done by passing the link to the `src` attribute of a html `<script>` tag. In other JavaScript files, a connection to a socket server can be set up and multiple methods to send and receive socket messages can be defined. In the following example the administrator sends a message to a socket server to stop a user. The user should be able to receive this socket message and thus automatically close the current application. But before the application is able to do so, the message should be defined on the server.

[10]`https://socket.io/`

[11]`https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.4/socket.io.js`

Sockets on the Server

After requiring the `socket.io` library, the http server needs to be transformed to a websocket server. Once this is done, multiple functions can be defined that can be called when a socket message is sent by a client to the server. A small example is given in Listing 4.6. In this case, the server can receive a message with title "`stop_app_user`". This will be a message been sent by the administrator. The server on its turn will send a message with the same name to all connected sockets in the system. When users are receiving the message, they will check if the `userId` variable inside the `data` object matches their userId.

```
// Import all the libraries needed
...
var socket = require('socket.io');
...
// transform http server into a socket server
var server = http.createServer(app);
var io = require('socket.io').listen(server);
server.listen(3000, function(){
 console.log('Server␣started␣on␣port␣3000');
});
...
// on socket connection, the messages become availble
io.on("connection", function(socket){
 socket.on("stop_app_user", function(data){
  io.emit('stop_app_user', data);
 });
});
```

Listing 4.6: WebSocket server with method to stop an application for a certain user

Sockets on the Client

There is not much needed to let sockets work on the different clients. Once the `socket.io` library is imported, the functions can be used in JavaScript files. In this particular example of blocking a certain user, the administrator and user should connect to the socket server. When the user selects an app, the administrator sees this in the admin tool. Then, the admin can select to kick the user out of the current application. In the JavaScript file, this is done as shown in Listing 4.7.

```
// global js file: connection to socket server
var socket = io.connect('');

// users.js file: stop user from using application
function confirm_stopping_app(user_id){
 $.ajax({
  url : '/users/'+user_id+'/stop',
  method : 'PUT',
  crossDomain : true,
  dataType : 'json',
  success : function(data){
   ...
   // First send ajax request to update JSON files. Once this is done, send
       socket message to server.
   socket.emit('stop_app_user', {userId : user_id});
  }
 });
}
```

Listing 4.7: Admin stop application for certain user

The socket message is now being sent to the server. As mentioned before, the server will pass the exact same message (‘`stop_app_users`’ with a `data` object containing the userId) to all clients. When a user is using an application, he will receive this message. The user will check if the userId in the message is equal to its userId, which is stored in a session variable upon login time. If this is the case, the user will be redirected to another page and the session variable containing the current application will be set to `false`.

```
socket.on('stop_app_user', function(data){
 if(data.userId == sessionStorage.getItem('user_id')){
  sessionStorage.setItem('active_app', false);
  window.location.replace("/hasys"); // redirect to homepage
 }
});
```

Listing 4.8: User can receive a message to stop the current application

With this simple example shown in Listing 4.8, the use of WebSockets and its power is easy to understand. The working for DUIs is very similar: the data object will contain a unique application id, together with some changes that have been made to the system. Users that are using this application will open the socket message and also perform the changes on their device.

4.2 Creating Mashups

The biggest and most important part of this thesis is the creation of mashups in the administrator tool. As mentioned in the ‘Solution’ section, flowcharts will play an important role. The first goal was to use Node-RED and to extend it so applications could be distributed. But after testing Node-RED by some unexperienced test users, they found it too difficult to use. One reason was the need to program a bit in JavaScript, something most of them did not have experience with. Next thing was to look around for some flowvchart libraries which were easy editable and easy to use for the end users. jsPlumb[12]

[12] `https://jsplumbtoolkit.com/`

and gojs[13] were tested, but the community editions worked a bit slow or did not offer exactly what was needed for this system. Therefore, a whole new flowchart program was created with techniques like svg to draw elements, and jQuery to connect them correctly and save them to the server.

4.2.1 Dragging Elements to the Canvas

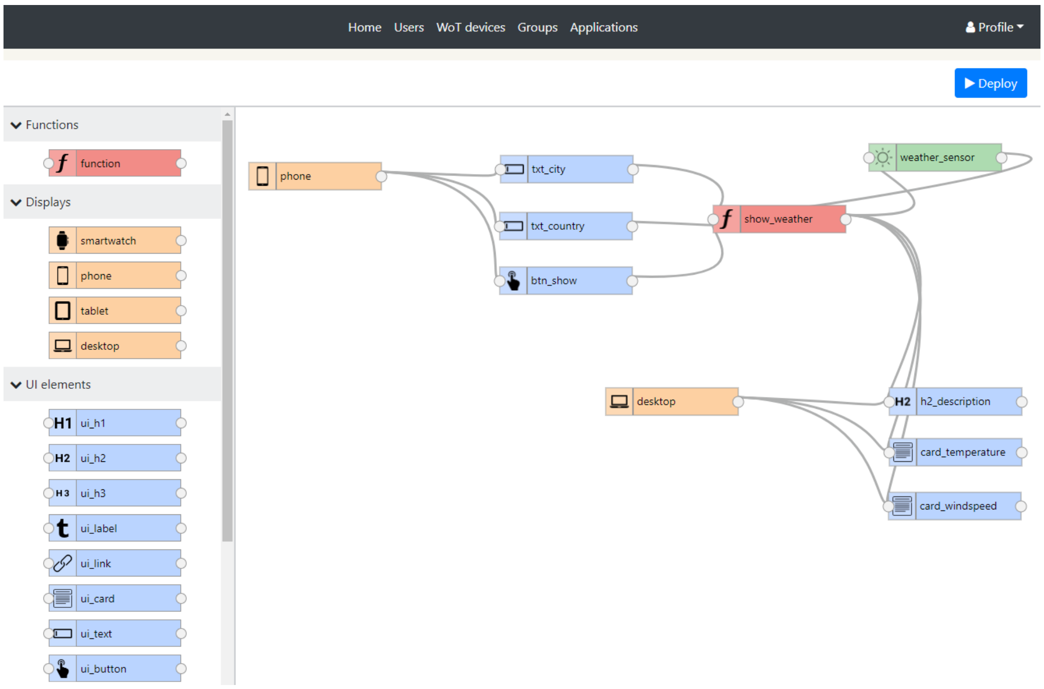


Figure 4.3: Constructing a mashup. Different UI elements are connected to a phone and a desktop. A function will show information about the weather in a city that is passed on the phone on the desktop

On the left side of the page a list of elements is given. All these elements are saved in the `elements.json` file and can be received by calling the REST API. Each element has the following properties:

- a category, for example ‘display’ or ‘ui’. In this way it gets listed underneath the correct accordion group.
- an icon and color to markup the node. Each category has its own color.

[13]`https://gojs.net`

- portIn: has the attribute `available` to specify that there is an inPort available for the node (this will be drawn on the left side of the node), and an attribute `connectable`, which is an array of other elements which can be connected to the inPort of this element. Some nodes, like a WoT device or UI element, will also stores setters. These will play an important role when constructing functions. If the portOut of a function node is connected to the portIn of a WoT device or UI element, it is possible to assign a new values to these elements using their setters. More information can be found in the subsection 'Constructing Functions' later in this chapter.

- portOut: similar to portIn, but this time for port that is situated on the right side of the node. The outPorts of UI elements, WoT devices or sensors can share their getters and events when connected to a function node. More information can again be found in the subsection 'Constructing Functions'.

- max: the maximum number of nodes of this type that can be drawn on the canvas. This is important for the display types, for example, only one desktop element should be allowed to be drawn on the canvas. Afterwards, the maximum number of desktop devices that can use the application at the same time can be specified when passing the details. At this time, it is not possible to define two different interfaces for two different desktops. Meaning each desktop will have the same interface and the same possibilities.

Once all elements are listed in the accordion on de left, they have to be draggable towards the central canvas. To drag an element, the library jQuery UI[14] is used, which makes it is easy to specify that some elements are draggable and to which area they can be dropped. If an element gets dropped onto the canvas, it gets transformed from an HTML element to an svg element. This is achieved in the following way: when an element gets dropped on the canvas, the program extracts its id and the x and y position where it is dropped. Before showing on the canvas, a modal will pop up asking the administrator for some extra information. In case of a textbox, the administrator has to pass a unique id, a text for the placeholder, and a value text. For each UI element in particular some widths have to be defined. For each type of device, a score going from 1 to 12 has to be entered. With this feature, it becomes possible to align UI elements differently on different devices. E.g.: when dragging a button to the canvas and giving a score of 6 desktop width

[14]`https://jqueryui.com/`

and 12 smartphone width, the button will show over the whole width of the page on a smartphone, and taking only the half of the screen when using the app on a desktop. If all validation checks succeed (in case of the textbox only a unique id is required), a function `send_elements_to_canvas()` will be called. This function expects the unique id and information about the element. In this way, it becomes possible to pass the correct information to the svg function and draw the desired elements on the canvas. The library jQuery SVG [15] is used to draw a svg element on the canvas using jQuery.

4.2.2 Connecting Elements

Placing elements on the canvas is one thing, but if they cannot be connected to each other they have no use at all. To connect elements, an inPort or outPort has to be selected. Once a port is selected, all the other ports on the canvas turn red or green. A green port means the current selected port can be connected to this port. This situation is depicted in Figure 4.4. A red port means that the selected port cannot connect to the selected port. In this way, it becomes impossible to create 'meaningless' applications. When a port gets selected, an array containing all the elements on the canvas is traversed. Their inPort or outPort properties will be checked to see if the selected port can connect. When the administrator clicks somewhere on the canvas else than a port, all the ports will be deselected and again be marked in their default color.

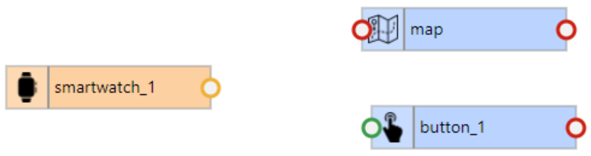
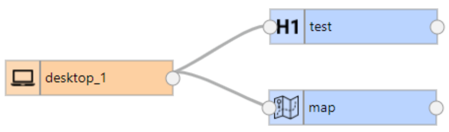
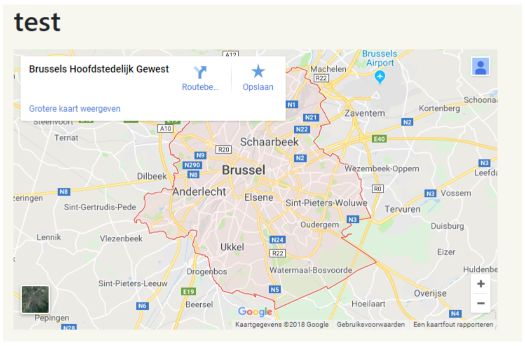


Figure 4.4: The outPort of a smartwatch element gets selected (orange). It is only possible to connect it to an inPort of a button (green). A map cannot be shown on a smartwatch (red). The outPorts of elements other than the selected one will always be disabled (red)

[15]http://keith-wood.name/svg.html



(a) Mashup tool



(b) Generated app

Figure 4.5: The `h1` element is placed above the `map` element (a). Therefore the title will also show above the map in the generated app (b)

In case of connecting UI elements to display elements, there is one important thing to notice. The order (top to bottom) UI elements are placed on the canvas, plays a big role in the layout of the final application. Placing a `h1` element above a `map` element will result in another layout than the other way around, as shown in Figure 4.5.

4.2.3 Constructing Functions

So far, it is only possible to show different elements. To interact with those elements, functions can be dragged to the canvas. When a function gets added, the administrator can only pass a unique function name. Constructing the function becomes possible when elements get connected to the inPort and outPort of the function element. Elements that get connected to the inPort of the function allow the function to get data and select an event. Elements that get connected to the outPort give the function the ability to perform actions on those elements.

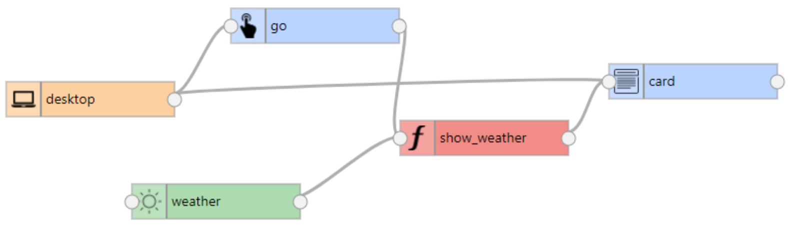
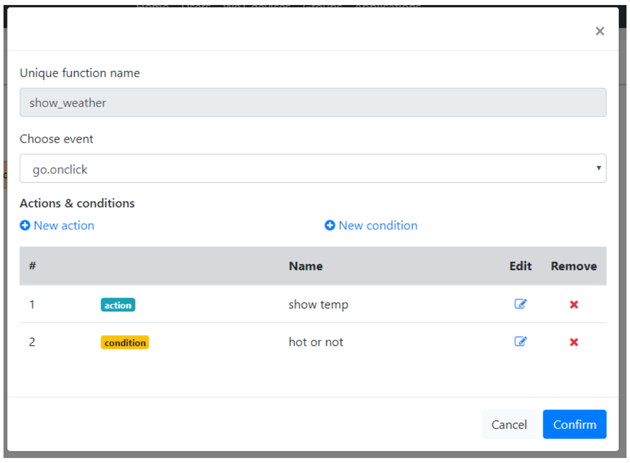


Figure 4.6: Mashup where a button and a weather sensor get connected to the inPort of a function, while a card gets connected to the outPort

In Figure 4.6, a button and a weather sensor get connected to the inPort, while a card (a Bootstrap element with a header, body and footer) gets connected to the outPort. This means the function can use the getters and events of the button and sensor, and the setter of the card. This way, it becomes possible to create a function where, when the users clicks on the button, the system will get the current weather and show the temperature in the card. Note that the event `page.onload` will always be available. Other events will be appended to the list when elements get connected to the inPort. While creating a function, the admin can develop actions and conditions that make part of this function. It is in these actions and conditions that getters and setters can be used. Figure 4.7 gives an overview of a particular function.


Unique function name: show_weather

Choose event: go.onclick

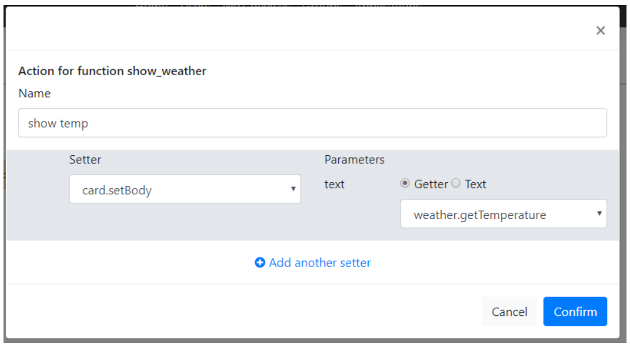
Actions & conditions

#		Name	Edit	Remove
1	action	show temp		
2	condition	hot or not		

Figure 4.7: Info about a function: a name, the event on which this function will get fired, and a list of actions and conditions that are performed when the function gets called

Actions

An action is in fact only a set of setters which expect zero or more parameters. In Figure 4.8 an action is defined where the body of the card is filled with the current temperature passed by the weather station. It is also possible to just pass a text value instead of the value of some getter.

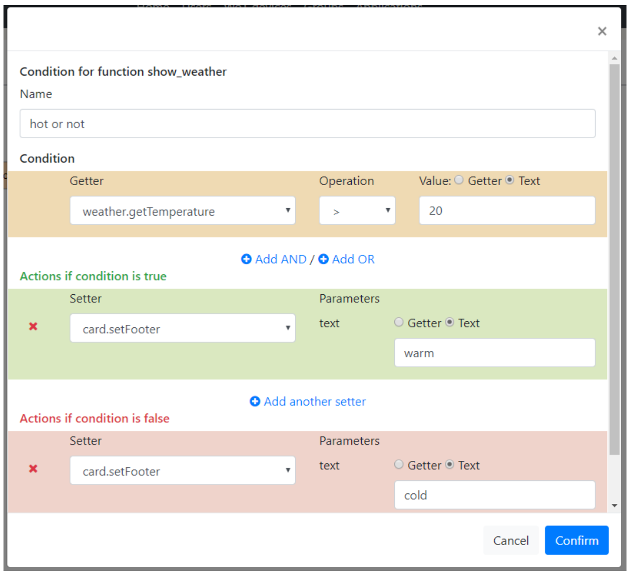

Action for function show_weather

Name

show temp

Setter	Parameters	
card.setBody	text	[✓ Getter] Text
		weather.getTemperature

Figure 4.8: Actions: setting the body of a card to the current temperature


Condition for function show_weather

Name

hot or not

Condition

Getter	Operation	Value: Getter [✓ Text]
weather.getTemperature	>	20

Actions if condition is true

Setter	Parameters	
card.setFooter	text	Getter [✓ Text]
		warm

Actions if condition is false

Setter	Parameters	
card.setFooter	text	Getter [✓ Text]
		cold

Figure 4.9: Condition: checking if the current temperature is above 20 degrees or not. Perform different actions based on the outcome of the condition

Conditions

Next to actions, it is also possible to create conditions. In these conditions, the getters can be used to perform checks on. Afterwards, different actions can be assigned to the case when the check holds, or when a false result is met. In Figure 4.9, a check is done if the current temperature is above 20 degrees. When this condition is met, the footer of the card will show the text ‘warm’. If the current temperature is below 20 degrees, the footer will show the text ‘cold’.

4.3 User App

4.3.1 Homescreen

Once a user is accepted by the system administrator, a list of all available applications for this specific user will be shown. An application is shown in this list if the user is connected to it by the system administrator AND if the device type of this user is present in the mashup. For example, if user1 is a tablet and allowed by the administrator to use a specific application, but in the mashup of this application no tablet element is present, the application will not be shown in the list. If in a mashup a specification is made were only a maximum number of users of a specific type can connect (e.g. only one smartphone user at a time can use application A), then this list item will be disabled and thus not allowing the user to select it.

Once a user selects an application, the `program.html` file will be loaded. Inside this HTML file, four important JavaScript files will be called: `engine.js`, `interpreter.js`, `condition_eval.js` and `setter_parameters.js`. In the next sections their functions will be explained in detail.

4.3.2 Engine.js: Drawing and Managing the Application

An application can be opened in two ways: or the user selects it from the list, or another device distributed its state to this device. In the first case, the application ID is passed in the url, together with the distribution status (which will be false in this case). In the second case, the same parameters are passed, only this time with the distribution status 'true'. Additional parameters `from` and `to` are passed to inform the system which user wanted to distribute its state, and to which user it was intended to distribute.

If the user opened the application by selecting it from the list, the application will just be drawn and values will be assigned as specified in the mashup. If the application got distributed by another user, the current values of elements on the page of the distributor will be asked for and sent back the user. The application will be drawn, but the values will now be the ones that were received from the distributor. The drawing process is the same in both cases. Wires of the mashup are investigated as shown in Listing 4.9. If the device type of the user is connected to UI elements, these UI elements will be stored in an array. Once the array is complete, it gets sorted by the y position of the UI element in the mashup, going for small to big. Afterwards, the elements are drawn on the page in this order: elements that appear higher in the mashup, will also be drawn at the top of the generated HTML page.

```
function draw_app(){
 // run through all wires
 $.each(app.mashup.wires, function(index, wire){
  // display (device) element will always be the start of a wire.
  // and always be connected to a UI~element (no other possibilities)
  if(wire.start == display){
   // store UI~element in array
   ui_elements.push(wire.end);
  }
 });
 // sort array based on y position
 ui_elements = sort_elements_by_y(ui_elements);
 for(ui in ui_elements){
  // draw UI~elements based on y position
  draw_ui_element(ui_elements[ui]);
 }
 // if all elements are on the page, assign functions to them (if applicable
    )
 assign_functions();
}
```

Listing 4.9: Drawing an application. Go through all the wires to get UI elements, afterwards sort them correctly on y position and draw them on the page.

Next to drawing the application, functions are assigned. All function elements of the mashup are investigated: the engine will check on which event of which element the function will be triggered. In this way, for example, a onclick function can be assigned to a previously drawn button UI element. Functions that are appended to a page load will be fired immediately. A concrete example and how functions will be interpreted is given in the next section.

Finally, the engine also listens to the socket server and handles incoming socket messages.

4.3.3 Interpreter.js: Handling Functions

Functions are assigned to elements in the following way: the JSON file stores the unique function name, the element where the function will be connected to, and an event of this element on which the function will be triggered. Next to this, an array with multiple actions and conditions is stored which will be performed once the function is triggered. An example can be a function `send_values`, with as element a button with id `send` and event `onclick`. The button will already been drawn by the engine. After putting all the elements on the screen, the engine will assign all functions. For this particular function, an attribute `onclick` having the value `do_function(send_values)` will be appended to an element with ID `send`.

Once this button is clicked, the function `do_function` with parameter `send_value` will be called in the `interpreter.js` file. Based on the function name (the parameter), all information of the function will be requested. All subfunctions of this function (actions and conditions) are performed in the order they are stored in the JSON file (and thus were defined in the mashup tool).

Performing Actions

Each action stores an element to perform the action to, a setter to alter an attribute of this element, and some parameters for this setter. If a setter has no parameters, to function `perform_setter` will be called immediately. If parameters have to passed, the function `do_setter_with_parameters` is performed in advance. Inside this function, checks are being made if a parameter is just text, or a getter. In case of a getter, the value of a specific UI element first has to asked for, or a AJAX request has to be made to a third party or WoT device to receive values. It is important that all AJAX requests are finished before actually calling the `perform_setter` function and assign all the values to the element.

Before calling the `perform_setter` function, a check will be made if the element is present on the current page. If this is the case, the function will be called. Inside this function, a switch with multiple possible cases is defined. The element, setter and optional parameters will be passed to this function. Based on the setter, the correct case is chosen. Inside this case, the correct values from the parameters array will be assigned to the correct attributes of the elements. E.g.: for a map element the setter `setPosition` can be passed with parameters `city` and `country`, resulting in a new position on the map that is drawn on the page. A global array containing all the current values is also updated to these new values.

After the check if the element is present on the current page or not, a message will be sent to the socket server, which on its turn will send the message to all devices that are currently using the application. Inside this socket message, the element, setter and parameters are passed. The devices will then call the function to perform the setter on the correct element and assign the right values to it. In this way, it becomes possible that a tablet can enter a city and a country on its screen, presses a button, parameters will be filled in and will be sent to a desktop device using a socket. This desktop device will perform the setter (e.g. changing the position of a map that is depicted on its screen) based on the parameters that were passed by the socket.

Performing Conditions

To interpret a condition, all available information provided by the JSON file is traversed. Each condition has a condition to check, a set of actions to perform when the conditions holds, and a set of actions when the condition fails (see Listing 4.10). The actions are performed in the same way as described above.

A simplification is made in the mashup tool: only AND, OR and subconditions are allowed. There is no possibility to add brackets. In this way, it is simple to divide the whole condition in multiple parts. Each subcondition is stored in the JSON file as depicted in listing below.

```
"conditions": [
 {
  "element": "philips_hue",
  "getter": "getIsOnLight1",
  "operator": "==",
  "valueType": "text",
  "valueText": "true"
 },
 {
  "combinator" : "AND",
  "element": "philips_hue",
  "getter": "getIsOnLight2",
  "operator": "==",
  "valueType": "select",
  "valueElement": "txtLight",
  "valueGetter" : "getValue"
 },
],
```

Listing 4.10: Condition having two subconditions: first one checks if the first light is on (standard value "true"), the other one checks if the second light has the same status as passed in a textbox. The AND combinator is used to combine both.

Subconditions will be stored in different array entries, depending on their combinator. The first subcondition does not have the combinator field and will automatically be stored in the first entry. In the listing, the first condition will check if the first light of the `philips_hue` element is true. The `valueType` field can have the value ‘text’ (a constant value) or ‘select’ (a variable value, depending on the element and getter that is given next). For the next conditions, the combinator will decided if the subcondition is placed in the same array entry, or if a new entry is created. If the combinator has the value AND, the subcondition will be placed in the same array entry. If the value of the combinator is OR, a new entry will be created and the subcondition will be assigned to this entry. The current entry is set to this new entry afterwards.

In the end, an array will exists with multiple entries. It now becomes possible to interpret the whole condition. For each entry, the following is done:

the first subcondition in this entry will be checked. The `condition_eval.js` file will be called to interpret this subcondition correctly. The outcome of this subcondition will be true or false. In case the outcome is false, the result of the whole current array entry will be false (logical AND). In case the outcome is true, the next subcondition of the current array entry will be investigated and the same procedure is done: a false result means going to the next array entry immediately, a true result will check the next subcondition of the same array entry.

If all array entries are checked, we have a new array with the same amount of entries, but now containing the values true or false. If all values are false, the whole end result will be false. If one value is true, the whole end result will be true (logical OR). Depending on this result, the right actions (stored in `actionsTrue` and `actionsFalse` in the JSON file) will be evaluated.

5 User Study

To test the usability of the system, a user study was performed. After a small explanation what the Web of Things and Distributed User Interfaces are, the participants got a small training of 10 minutes on how to construct a mashup. The basic features were explained (e.g. connecting UI elements to different kind of devices, constructing functions). There was no time limit set. When their application was ready, the users got to fill in a questionnaire and give at most three positive and negative comments about the system. The Computer System Usability Questionnaire [39] was used in order to answer the questions.

5.1 Setting up the Experiment

5.1.1 Participants

Before actually doing the experiment, several participants who would like to test the system needed to be found. Once this was done, they were all invited to come perform the experiment. Due to the availability of the participants, multiple sessions were held on different days. Each of them got a small training on how to use the mashup tool and constructed the defined application afterwards.

In total 14 different people did the experiment. Participants of this study had no or limited programming experience. However, they all had some knowledge about flowchart diagrams and how they work. The age range of the participants goes from 20 to 50 years old. Ten men and four women participated. All of them are interested in new technologies, but a small explanation on what Distributed User Interfaces are and how physical things can be controlled over the Internet using a REST API was given.

5.1.2 Setup

The setup of this experiment consisted of one laptop that could be used to create the desired mashup. A Philips Hue was present as well to be able to control a smart physical device. The environment of this test was a simple living room. The participants used their own smartphone and the available laptop and tablet to test the generated application. To make things easier, a small documentation for the Philips Hue with the most important urls and functions was given in order to speed things up and do not let the users get lost in the Philips Hue API documentation.

5.1.3 Training

Before actually creating the app, each participant got a small training from about 10 minutes. This training consisted of a PowerPoint presentation with screenshots of the mashup tool. First, users got to see how to add a WoT device and connect some functions to it. Afterwards, the creation of an application where a smartphone user could control the city of which the weather data was shown on a desktop was explained. In this way, the participants got to see how to connect different UI elements to different devices, how to create functions and how to use third party data over a REST API. Given this training, the users should be able to create an application on their own.

5.2 Creating the Mashup

After the training, the participants got an assignment to create a mashup involving a WoT device and different display devices. Once the creation of this mashup was done, the participants had to connect their own smartphone to the system and accept it as an administrator. The assignment was specified as follows: "*The participant should be able to create a mashup that involves all following aspects. A mobile phone and tablet will be used to control a light from the Philips Hue. A button having the text 'Light on' or 'Light off' will* *be shown. Depending on the device type, the button will be shown over the full width of the page (in case of a mobile phone), or on only half the page (in case of the tablet). On a desktop a <h1> element will be presented having the text 'The light is on' or 'The light is out', depending on the current status of the lightbulb. The user should first connect the Philips Hue to the system and add some getters and setters to it to be able to ask for the current status of the light, and to change this status afterwards. Once this is done, the mashup can be created. The user should create two functions:* ***`light_on_load`*** *and* ***`switch_light`****. The first function should be triggered when the page is loaded, the second one is fired after a specific button is clicked. Both functions will have the ability to show the correct values (text on the button and text of the <h1> tag), depending on the current status of the light."*.

No time limit was set to create this mashup. Participants could play around and test as much as needed. After the desired application was finished, the questionnaire had to be filled in.

5.3 Questionnaire Results

As mentioned before, the Computer System Usability Questionnaire (CSUQ) [39] was used to record the usability of the system. The participants had to fill in the questionnaire that could be found on the Internet[1]. It is worth noticing that this questionnaire is slightly different than the one described in the paper. Where in the paper the Likert scale goes from 1 to 7 with 1 being 'Strongly agree' and 7 'Strongly disagree', the webpage works the other way around. Meaning that a higher number means a better result, and not a worse one like in the paper. At the bottom of the page, the participants had the possibility to fill in at most three positive and negative comments about the system if they like.

Results from the CSUQ can be divided into four separate groups: the overall satisfaction score (OVERALL), the system usefulness (SYSUSE), the information quality (INFOQUAL) and the interface quality (INTERQUAL). In Table 5.1 an overview of which questions belong to which subsets is given. As the Likert scale goes from 1 to 7, with 7 being the best score, it can be said that a mean score above 3.5 can be considered as good. Next the mean and standard deviation of each subset are calculated. The standard deviation will tell how widely the scores of different participants are spread. A high standard deviation (here set to any value above 2) will mean that the data is not consistent. Afterwards, some positive and negative remarks from the participants are listed that can be used to improve the system in the future.

[1]`http://garyperlman.com/quest/quest.cgi`

Subset	Questions
OVERALL	1 to 19
SYSUSE	1 to 8
INFOQUAL	9 to 15
INTERQUAL	16 to 18

Table 5.1: Overview of subsets of the Computer System Usability Questionnaire

5.3.1 Quantitative Data Analysis

The results of the CSUQ can be divided in quantitative data (the 19 questions that had to be answered by assigning a score from 1 to 7 to it) and qualitative data (giving some positive and negative remarks afterwards). In this section, the quantitative data will be analyzed. Table 5.1 shows us the data can be divided into four subparts. Passing the data to a spreadsheet and perform calculations on it, gives us the results that are depicted in Figure 5.1.

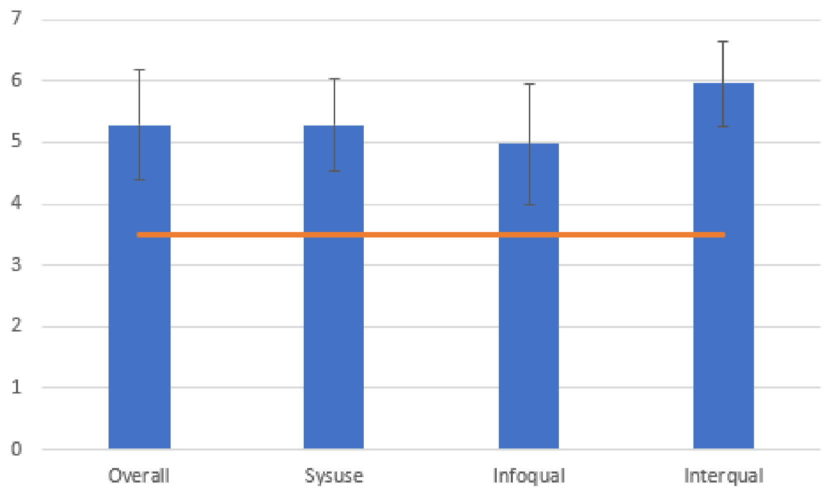


Figure 5.1: Bar chart depicting the means of each subset, together with the standard deviation, shown as error bars. The orange horizontal line shows the benchmark line of 3.5

Overall, the system has a positive evaluation with a mean of 5.28 and a standard deviation of 0.89, meaning that the data is not widely spread and the participants more or less had the same opinion about the system. Averaging the scores for the usage of the system resulted in the exact same mean as the overall mean. High scores within this subset were given to questions 3 and 8 (effectively completing the work and becoming quickly productive), a lower average score was given to question 4 (able to complete the work quickly). In a small interview afterwards, participants said they found it a bit hard at first, but once they knew the system things seemed to be more easy and they had a happy feeling when they saw the working 'advanced' application they created. Learning a programming language and different technologies would have cost them more time than it did now, but first playing around with the tool and discovering it before actually creating the mashup successfully gave some of them the feeling it took too long.

The information provided by the system (INFOQUAL) received an average score of 4.97, which is the lowest of all subsets. The standard deviation of 0.99 shows most participants agreed with each other. Question 9 (clear error messages) scores high with a mean of 6, but lower scores were given to the other questions. People expected more on-screen information and said it maybe would be more difficult to create the mashup if no 10 minute training was given before the start. Providing a tutorial could solve this problem in the future.

The last subset, the interface quality, received the highest average score of 5.95, with the lowest standard deviation (0.69). No real difference between the different means within this subset exists, all of them score very well. Users like the interface of the system and it gives them the opportunity to do the things they expect it to do, namely adding different kind of devices to the system and make applications to let them communicate with each other.

To conclude this section, it is worth mentioning that on average the users created the whole application in 15 to 20 minutes. Adding a WoT device to the system (given the simplified API documentation about the Philips Hue) was done very quickly (around 5 minutes), together with creating the different UIs for the different devices (also around 5 minutes). The time that was left was spent on creating the two functions.

5.3.2 Qualitative Data Analysis

After filling in the questions, the users could also pass some positive and negative comments about the system. Many of them liked the interface and found it very easy to find what they were looking for. The mashup tool itself was very clear (elements that can be draggged to the canvas on the left, a big canvas, a deploy button) and they liked using it. The error messages that were given (passing a unique ID, a required value, ...) gave them hints on what they were doing wrong. Assigning a red or green color to ports when one port is selected let them quickly see which elements were connectable to each other. Some of them say it maybe would be better that people actually could drag a line between ports (holding the left mouse button), instead of just selecting two ports, because they were used to this feeling when creating a flowchart. Also the generated wires between two ports could maybe be drawn in a different way, so it is always clear which wire belongs to which inPort and outPort. At this moment, when many wires exist in a mashup, it is not that clear because they get ‘intertwined’. Using a professional flowchart or graph library in the future may overcome this problem.

As already briefly mentioned in the previous subsection, some participants said it might be hard (or at least harder) to construct the mashup when no training was given. They do not really like the trial and error approach a lot when creating something. Especially because they are new in controlling these technologies and are worried to ‘break’ something. The suggestion was made to provide an info button on the mashup screen where people could find a tutorial. This is a very good remark and can be considered when extending this mashup tool.

To end with, the most positive remark is the one that most participants were happy that they could create an advanced application without the knowledge of too much technological details. With a good documentation provided by the mashup tool and by WoT device developers, they would love to discover and create new things.

6 Future Work

Limited research to the combination of mashup tools, Distributed User Interfaces and the Web of Things existed when starting this thesis. Therefore, the developed tool can be used as a starting point and many extensions are possible.

The User Study revealed that the users were overall happy with the system, but improvements could be made. One thing that was mentioned, was a clear tutorial that should be available for the unexperienced administrators and creators of the mashup. A small information icon can be added on top of the page with all the important information given, for example, how to connect UI elements to a display device and how to create functions. A further improvement could be the integration of a real step-by-step tutorial where users learn to perform all the important actions.

As the mashup tool itself was built from scratch improvements can be made. The user study showed when multiple wires are present on the canvas, it sometimes looked like they were intertwined and it was not really clear which wire was connected to which elements. One option is to use a third party API to overcome this problem.

Until now, there is no real possibility to debug the program, other than deploy it and use different devices to see what happens. In the future, a debug tool could be added where the application gets simulated on some virtual devices. This way the administrator should not always deploy the application and thus make an incomplete application available for the end users. In addition to this, a visualisation of how UI elements align on the different devices can be given. Maybe even with the possibility to edit the UI elements directly in this visualisation and extend the usability of the mashup tool. Further improvements could be to apply CSS styles to the different elements. At this moment, a button will always have the blue colour, and a label will always have the same font size and style. Adding the option to style these elements gives the system much more possibilities.

Built-in third party APIs are limited to the OpenWeatherMap API[1] and the Wikipedia API[2]. Multiple other options exist and can in the future be offered to the mashup creators. Likewise, not all possible HTML UI elements are integrated in the current mashup tool. E.g. it is currently not possible to show images, but this can easily be integrated in future versions of this system.

Physical things that follow the Web of Things approach can easily be integrated into the system by just adding them on the correct page. Meaning that more Web of Things devices will develop automatically new options for this mashup tool. However, at this time it is not possible to handle arrays. For example: it is not possible to print all the available lights of the Philips Hue on one page. Editing the mashup tool, as well as the interpreter on the user side will overcome this shortcoming.

On the user side, an optimization can be made where the results of REST API calls are stored in the local storage. At this time, if a function is constructed where the temperature and a description of the weather in a certain city is asked, the system will make two requests to the same API: one time to get the temperature, and one time for the description. Storing the JSON result after the first call will speed up the program (not needing to wait for a second AJAX call to be completed).

To end with, the function element of the mashup tool can be extended. For now, it is only possible to perform some actions (setting values of different elements) and conditions (getting values of certain elements and check if they fulfil a condition). Giving the administrators possibilities to insert a switch operator, or make a loop are some examples that can be integrated in the system. However, one should always keep in mind to keep it as simple as possible and make the mashup tool available for all kind of users: programmers and non-programmers.

[1] `https://openweathermap.org/`
[2] `https://www.mediawiki.org/wiki/REST_API/`

7 Conclusion

The goal of this thesis was to create a mashup tool to easily develop Distributed Physical-Digital User Interfaces. Before the development of the system started, related work was investigated. The related work showed that mashup tools for both Distributed User Interfaces and the Internet of Things exist, but none of them integrates both. Most existing mashup tools used a flowchart interface to create new applications. Therefore, this approach is also used in this thesis. Given the related work, important guidelines about Distributed User Interfaces and the Internet of Things were discovered and paid attention to while creating the system.

A general solution for the problem was defined after the literature study, keeping in mind the positive and negative aspects from previously created systems. The ultimate goal was to solve the problem of existing mashup tools. They exist for Distributed User Interfaces and for the Internet of Things, but not for a combination between them. Different user classes were defined: an administrator who creates the different mashups and controls other users of the system, which will use the created applications. Blocking users and applications, kicking users out of sessions, allowing a limited number of users to the system: a complete management of as well users as applications should be provided. A scenario with personas was defined in order to have a feeling what kind of users would use the system and what they would do in several cases. The design process defined by Moore [47] was followed to create the

User Interface. Usability requirements were set, user object models and tasks models defined, and a style guide specified. Afterwards, the iterative design process was started, using different kinds of mockups to show to the potential users. Finally, the system was developed.

Web technologies were used to create two applications: the mashup tool where an administrator creates different kinds of mashups and can control users, devices and applications, and an application for the users where they could select a generated application to use. HTML5, CSS3, jQuery, Express.js and WebSockets are the main technologies that were used to construct the system.

The final system offers the possibility to create mashups where an application can be distributed over multiple device types, can control and ask data from physical things, as well can ask third parties like weather stations or Wikipedia for data. The administrator has full control and can block applications, stop users from using a specific application, allow users to connect to the system, or block them for a period of time. Users can choose to select a certain application that is made available for use by the administrator. Each application can also be distributed to other devices by the user.

The user study showed that overall, the users were happy with the system. A small training of 10 minutes was enough to complete a predefined task successfully in 15 to 20 minutes. Of course, not all was perfect and some remarks were given. In the Future Work chapter, a solution for these remarks is proposed, together with possible extensions that can be made to the system in the future. As no specific mashup tool existed to create Distributed Physical-Digital User Interfaces, this tool can be seen as a good prototype that can be heavily extended to offer even more possibilities to the end users.

A
Appendix

A.1 CTTs

A.1.1 Manage User Request

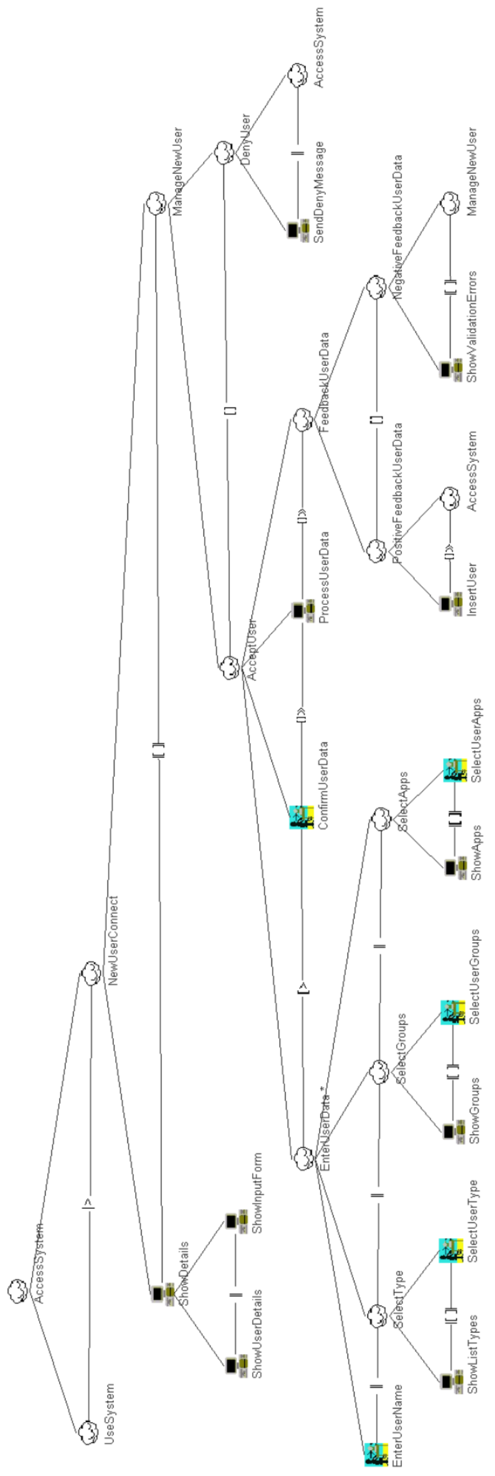


Figure A.1: New user tries to connect to the system

A.1.2 Using the system

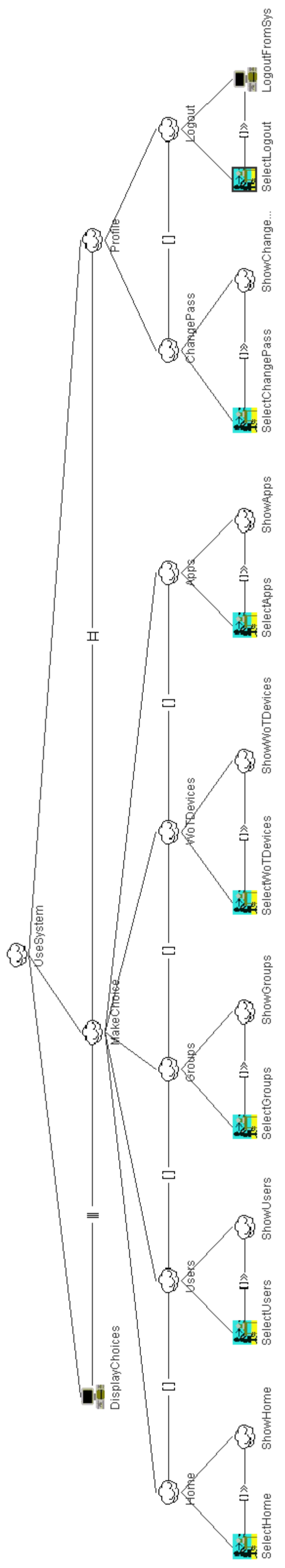


Figure A.2: Admin has several choices he can make, or he can perform some profile actions

Homescreen - User Using a Live App

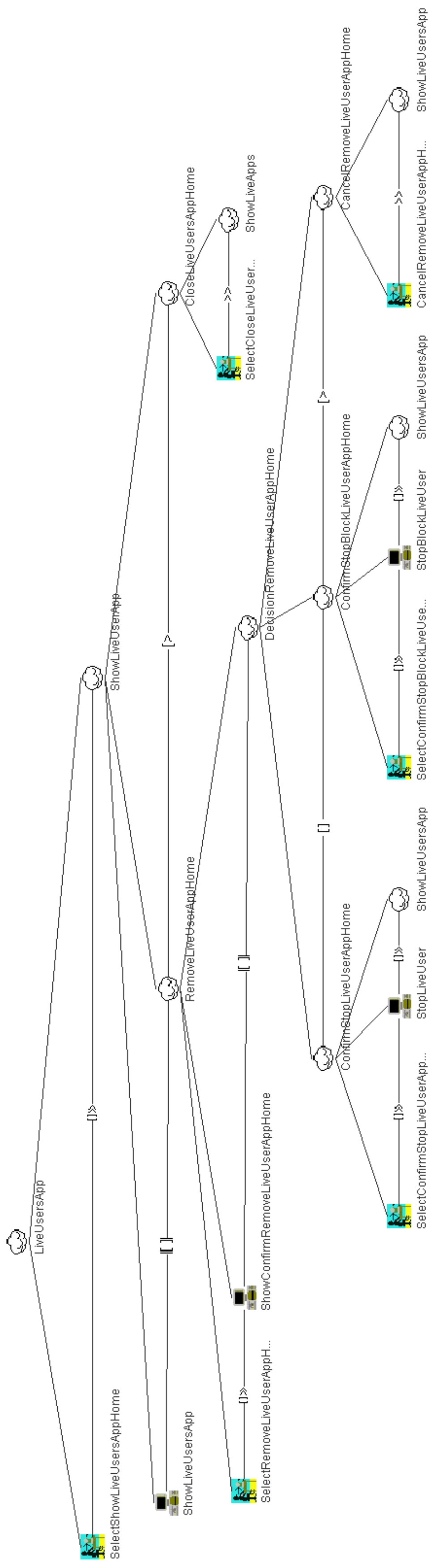

LiveUsersApp
SelectShowLiveUsersAppHome
ShowLiveUserApp
ShowLiveUsersApp
RemoveLiveUserAppHome
CloseLiveUsersAppHome
SelectCloseLiveUser...
ShowLiveApps
SelectRemoveLiveUserAppH...
ShowConfirmRemoveLiveUserAppHome
DecisionRemoveLiveUserAppHome
CancelRemoveLiveUserAppHome
CancelRemoveLiveUserAppH...
ShowLiveUsersApp
ConfirmStopBlockLiveUserAppHome
SelectConfirmStopBlockLiveUse...
StopBlockLiveUser
ShowLiveUsersApp
ConfirmStopLiveUserAppHome
SelectConfirmStopLiveUserApp...
StopLiveUser
ShowLiveUsersApp

Figure A.3: Showing an overview of users using a live application. Users can be kicked out of the session

Homescreen - Live Users

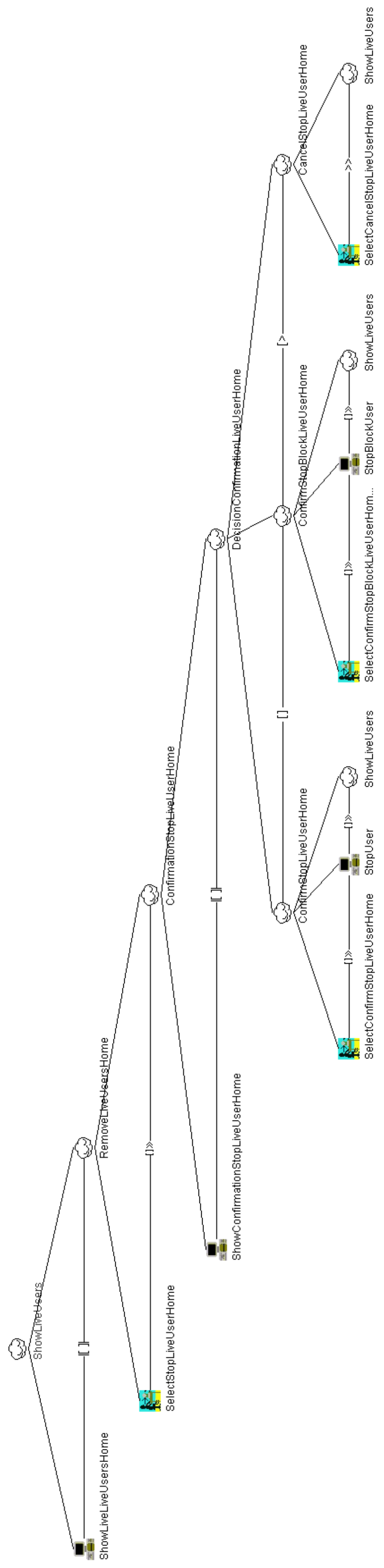


Figure A.4: Overview of all live users. Users can be stopped and blocked if wanted

Users - Show Groups

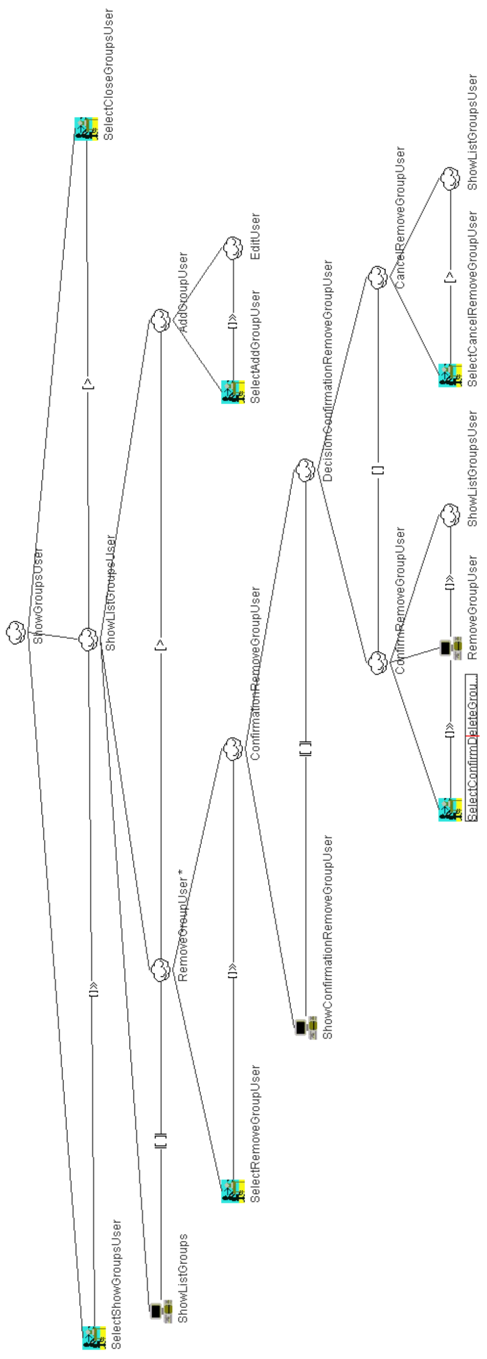


Figure A.5: Showing a list of groups the user is connected to. Connect new groups or remove existing ones

Apps - Mashup

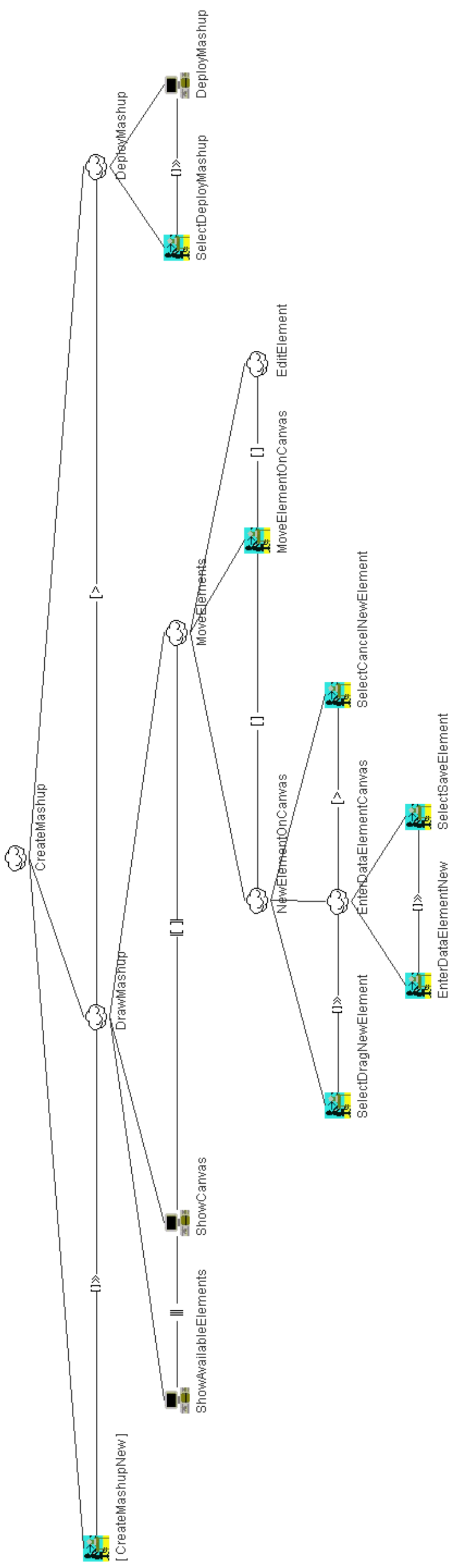


Figure A.6: Constructing the app with the mashup tool

Using an Application

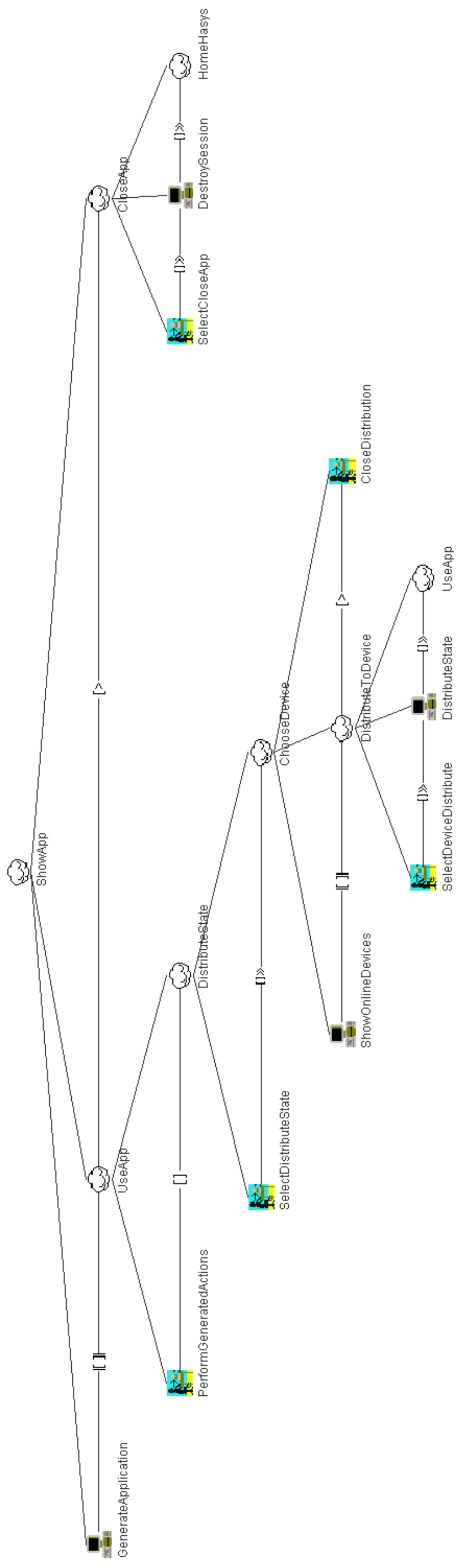


Figure A.7: Using a certain application with possibility to distribute state and close the app

A.2 ORM Diagram

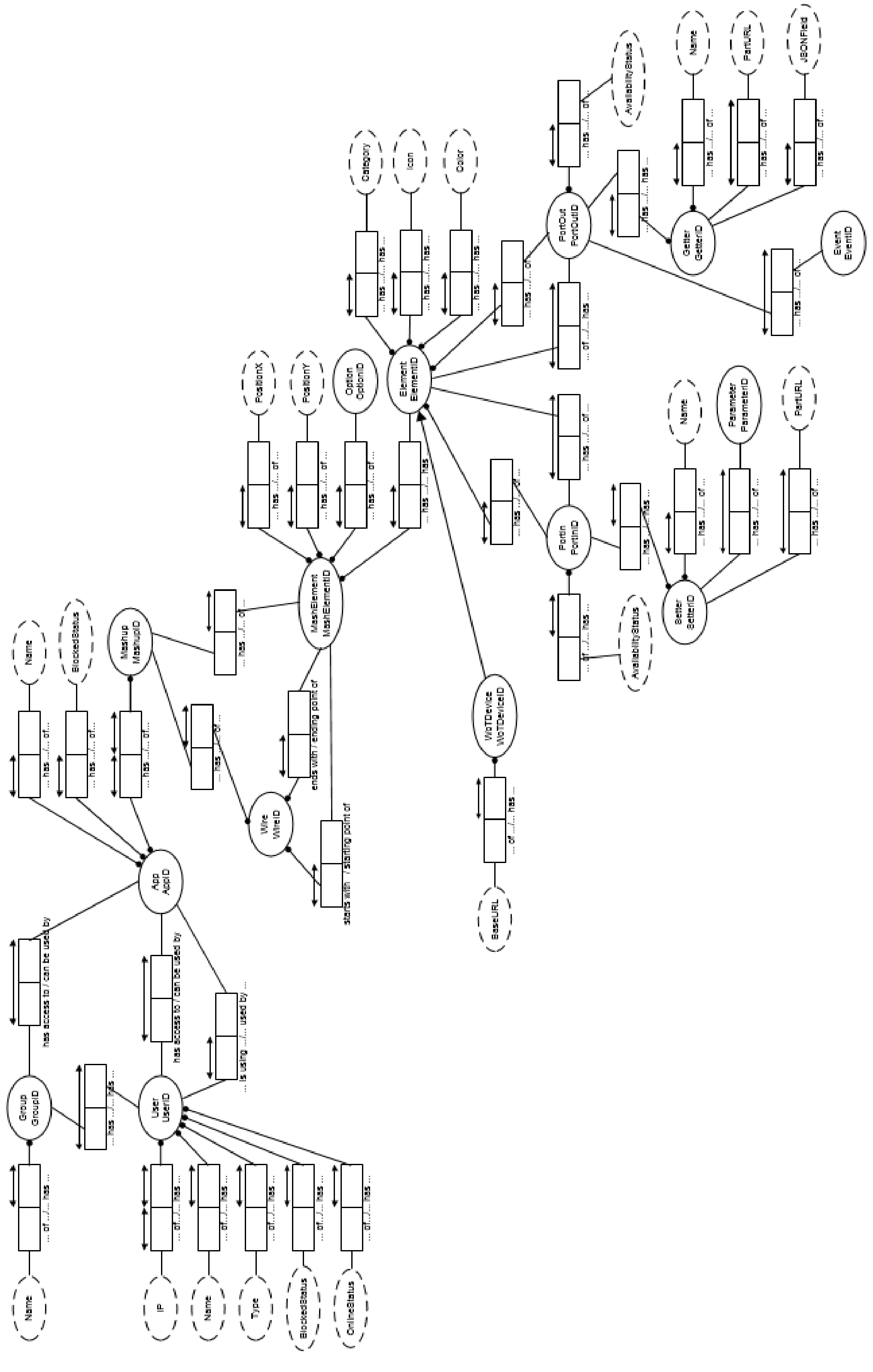
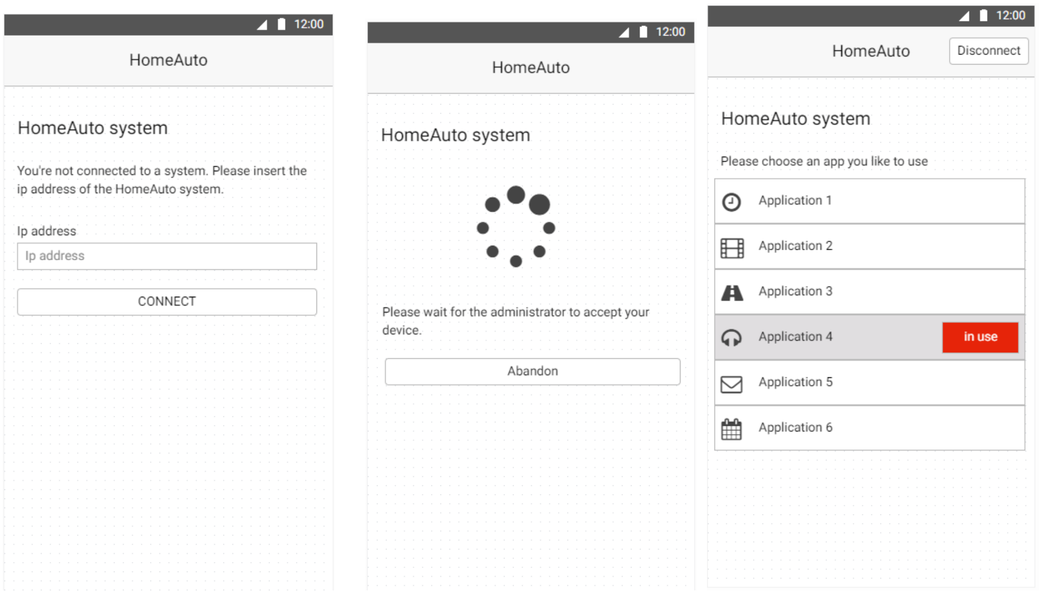


Figure A.8: Complete ORM diagram.

A.3 Mockups

A.3.1 User


12:00

HomeAuto

HomeAuto system

You're not connected to a system. Please insert the ip address of the HomeAuto system.

Ip address

Ip address

CONNECT

12:00

HomeAuto

HomeAuto system

Please wait for the administrator to accept your device.

Abandon

12:00

HomeAuto Disconnect

HomeAuto system

Please choose an app you like to use

Application 1

Application 2

Application 3

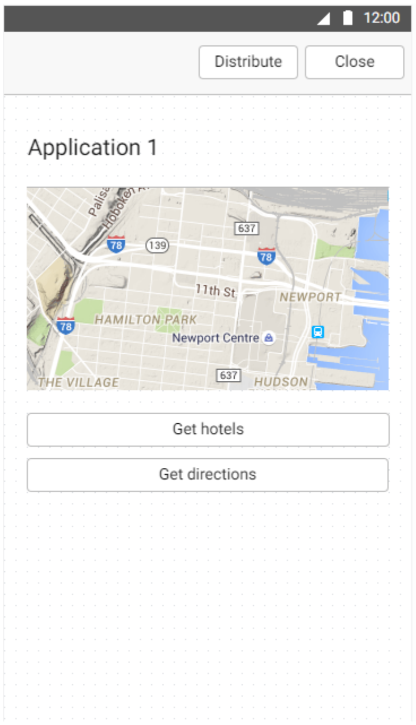
Application 4 in use

Application 5

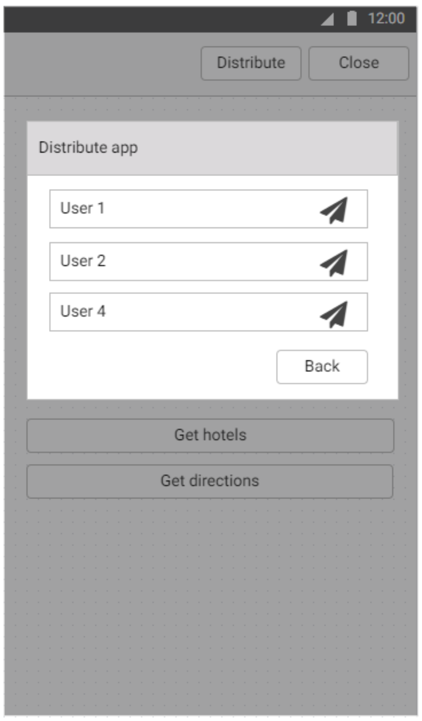
Application 6

(a) Connect to a system (b) Waiting for approval (c) All accessible apps

Figure A.9: The three mockups above show the screens where a user is not connected to the system yet and has to pass the right IP address (a). Once a request is sent to the server, a screen will be shown where the user has to wait for a response, but can abandon the request if desired (b). If the user gets accepted, all accessible apps are shown. If the number of users of a certain device type is reached for an app, the app will be unavailable for the user.



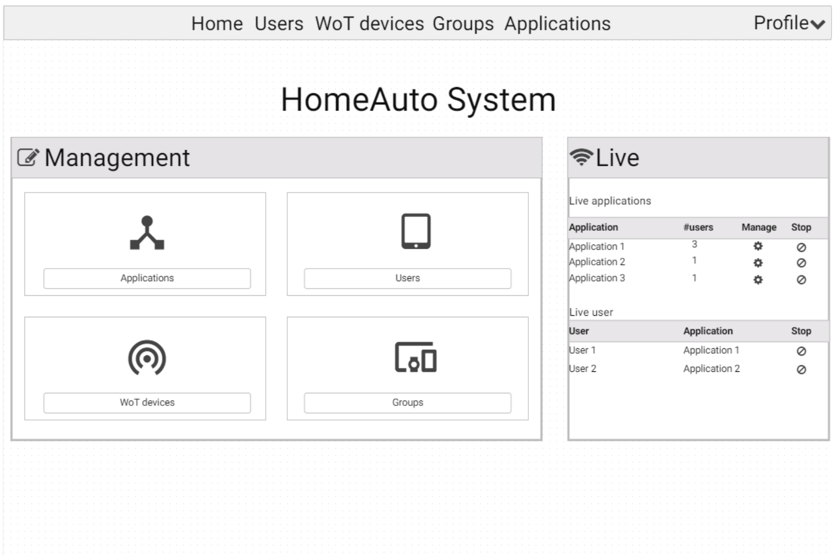
(a) Accessing an app



(b) Distributing current status

Figure A.10: Once an app is selected, it will be generated on the screen (a). If desired, the current status can be distributed (pressing the button on top), or stopped (also on top). In case of distributing an app, a box will pop up to select the device to distribute to.

A.3.2 Admin



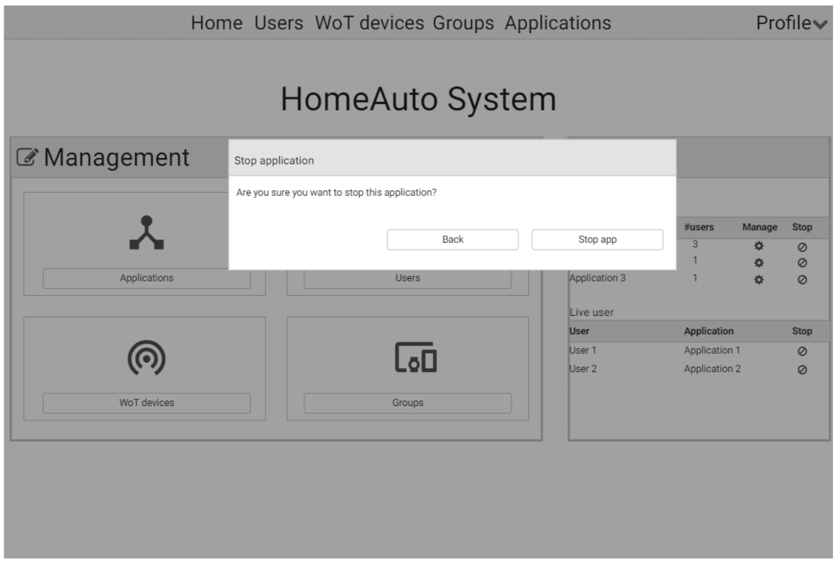
Live applications

Application	#users	Manage	Stop
Application 1	3	⚙	⊘
Application 2	1	⚙	⊘
Application 3	1	⚙	⊘

Live user

User	Application	Stop
User 1	Application 1	⊘
User 2	Application 2	⊘

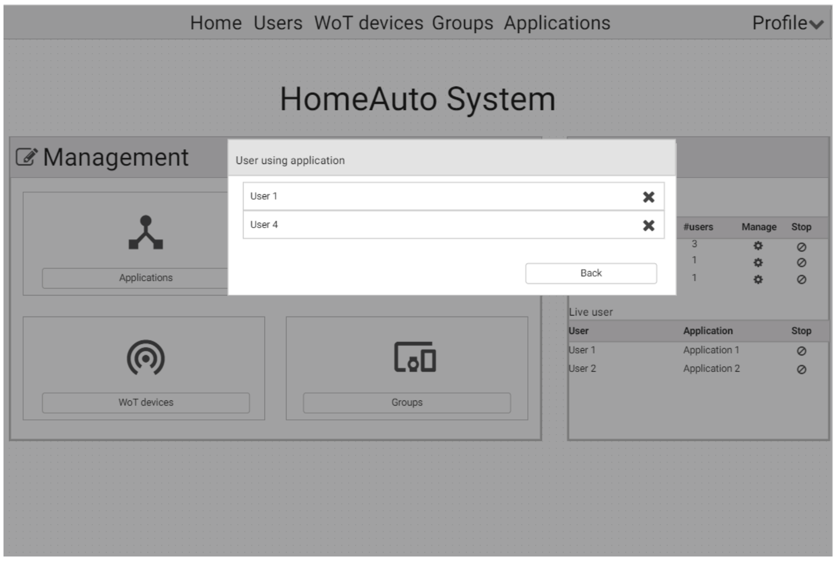
Figure A.11: Homepage administrator. Options to navigate to other pages, or to control live applications and online users.



Stop application

Are you sure you want to stop this application?

Figure A.12: Stopping a live application, a confirmation is being asked. The same action can be performed on the Applications page.



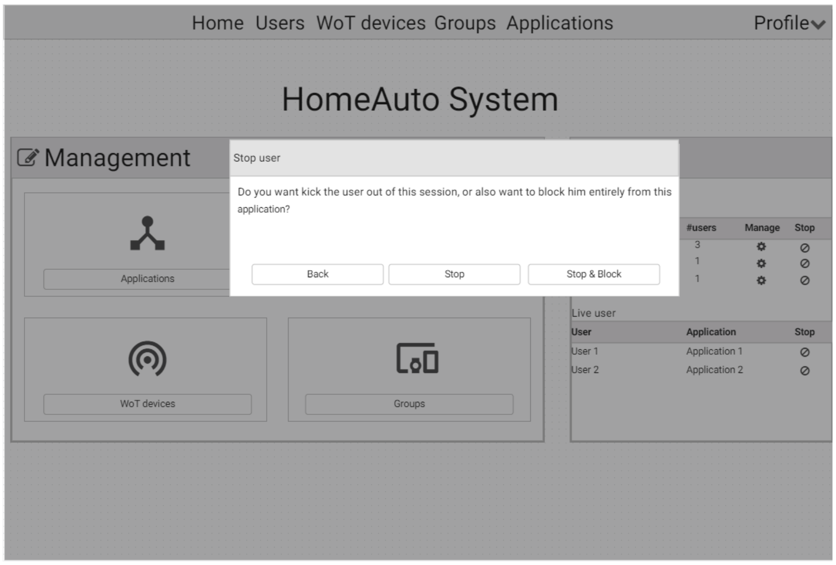
User 1 ✖
User 4 ✖

#users	Manage	Stop
3	⚙	⊘
1	⚙	⊘
1	⚙	⊘

Live user

User	Application	Stop
User 1	Application 1	⊘
User 2	Application 2	⊘

Figure A.13: Showing an overview of all users that are currently using a certain app. Users can be kicked out of the session. The same action can be performed on the Applications page.



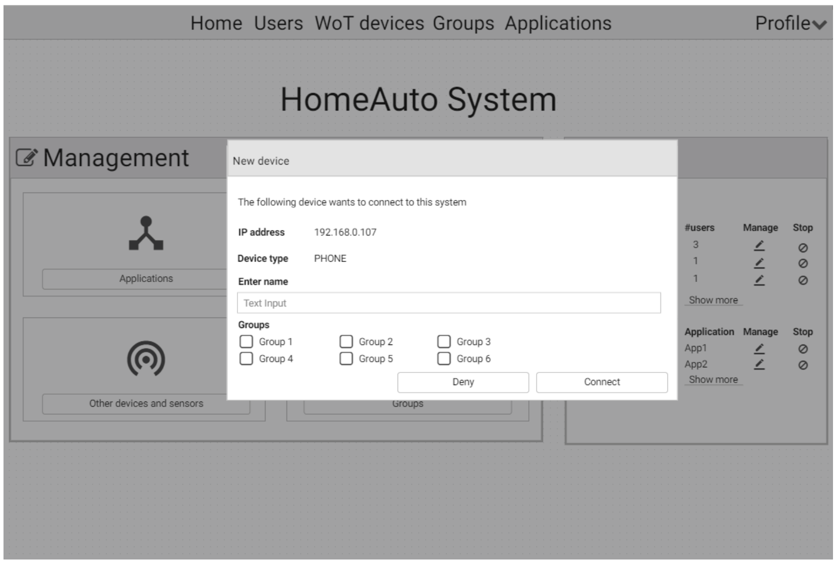
Do you want kick the user out of this session, or also want to block him entirely from this application?

#users	Manage	Stop
3	⚙	⊘
1	⚙	⊘
1	⚙	⊘

Live user

User	Application	Stop
User 1	Application 1	⊘
User 2	Application 2	⊘

Figure A.14: Kicking a user out of an application. Confirmation is being asked to just kick the user this time, or block the user completely from using the app ever again. A back button is available to cancel the operation.


Home Users WoT devices Groups Applications
Profile
HomeAuto System
Management
Applications
Other devices and sensors
Groups
Deny
Connect

New device

The following device wants to connect to this system

IP address	192.168.0.107
Device type	PHONE

Enter name

Text Input

Groups

☐ Group 1 ☐ Group 2 ☐ Group 3
☐ Group 4 ☐ Group 5 ☐ Group 6

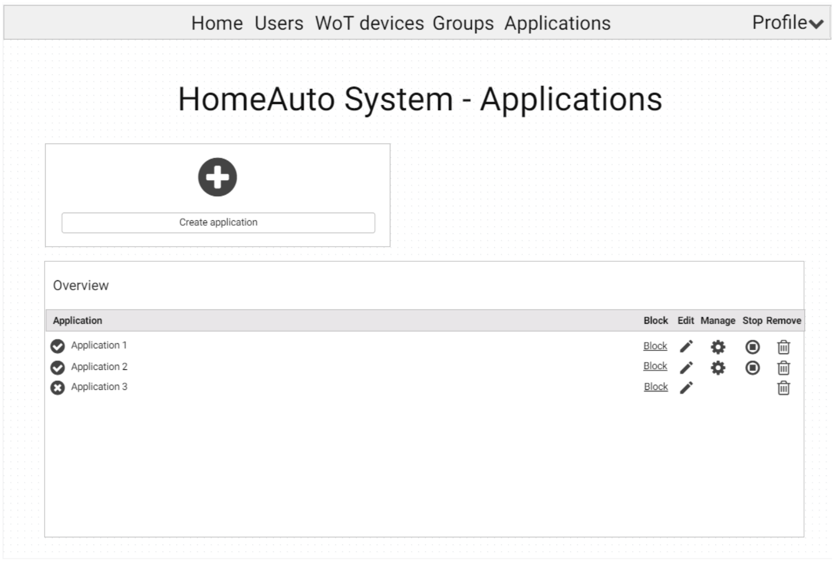
#users	Manage	Stop
3	✎	⊘
1	✎	⊘
1	✎	⊘

Show more

Application	Manage	Stop
App1	✎	⊘
App2	✎	⊘

Show more

Figure A.15: New user tries to connect to the system. On every possible page, a modal will pop up. Showing all the info and asking for some values. The admin can accept or deny the request.


Home Users WoT devices Groups Applications
Profile
HomeAuto System - Applications
Create application

Overview

Application	Block	Edit	Manage	Stop	Remove
✓ Application 1	Block	✎	⚙	⏹	🗑
✓ Application 2	Block	✎	⚙	⏹	🗑
✗ Application 3	Block	✎			🗑

Figure A.16: Overview page of all applications. Information if an app is live or not. Live apps can be stopped or managed (kicking live users out of the app, or block them from using the app again). Links to delete or update an app, or to create a new one are provided.

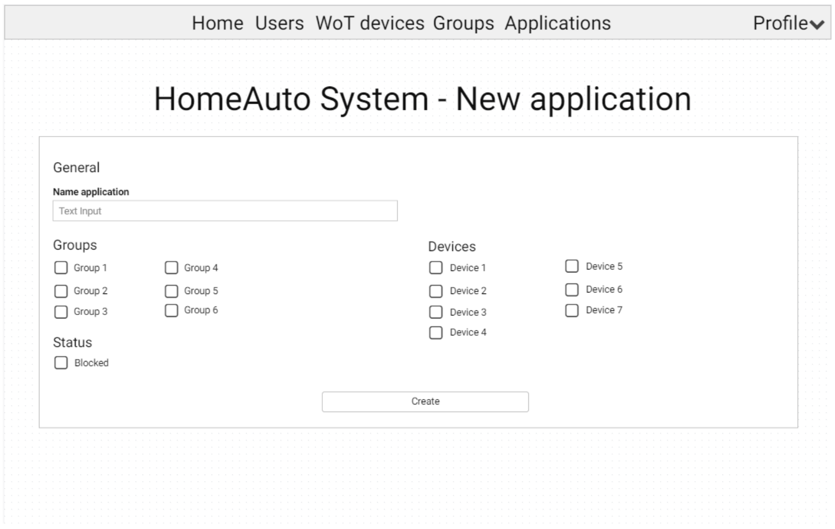


Figure A.17: Creating a new application. Basic information exists of a name, connected groups and users, and a block status (will the app already be available for the users or not?).

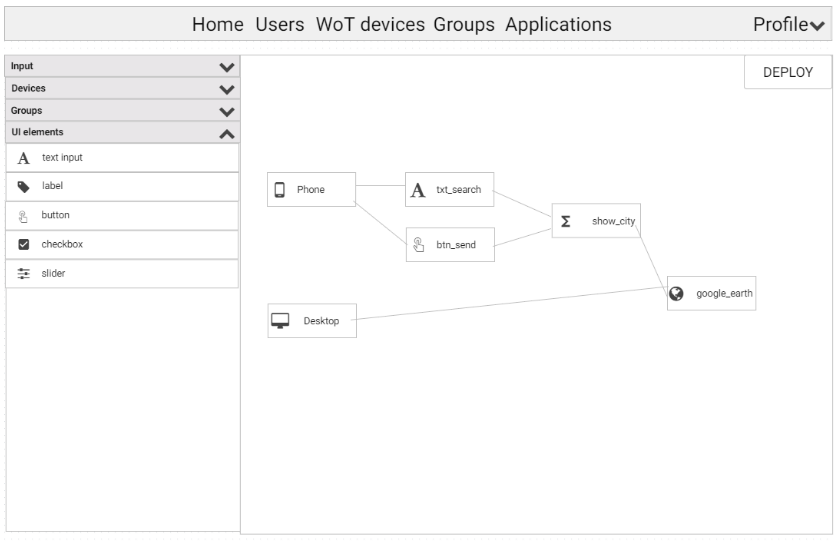


Figure A.18: After passing the basic app info, a mashup can be created. Elements on the left can be dragged to the canvas on the right and be connected to each other.

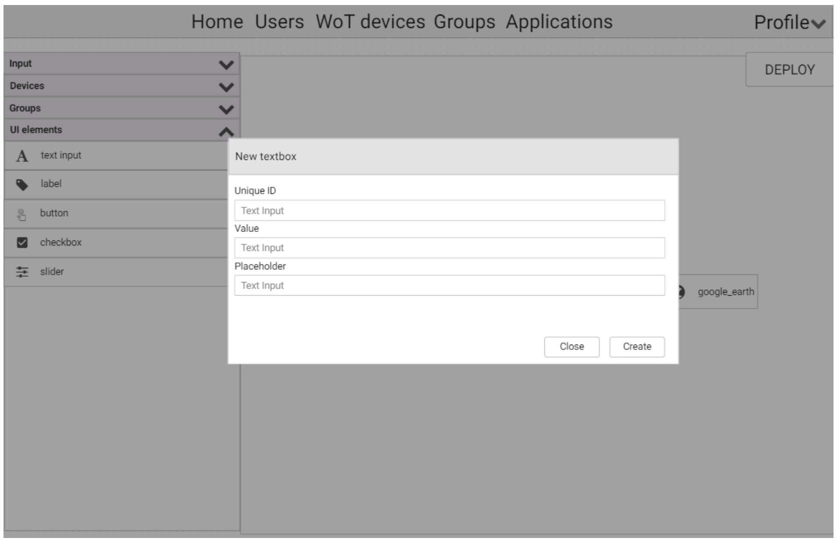
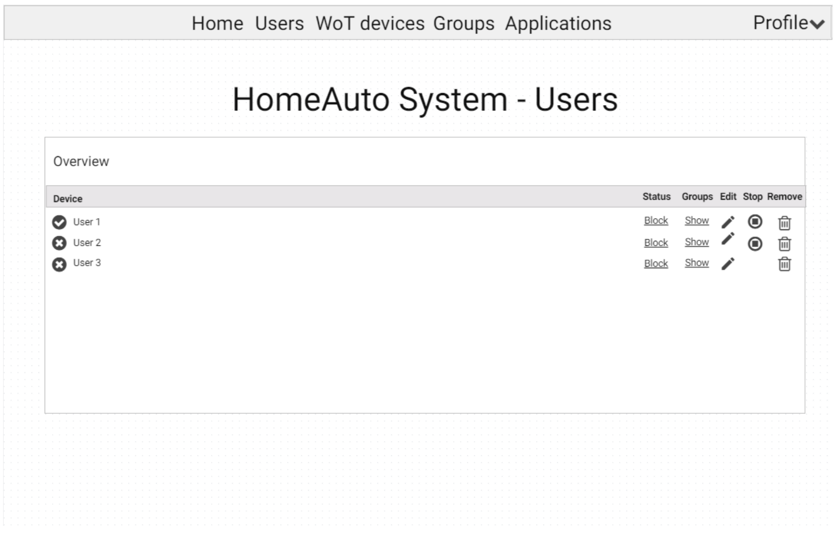


Figure A.19: A new element is dropped on the canvas. In this case, a textbox is dropped. An ID is required, also the value and placeholder of the textbox can be passed if wanted. This modal box will look different for each type of element that is dropped on the canvas.



HomeAuto System - Users

Overview

Device	Status	Groups	Edit	Stop	Remove
User 1	Block	Show			
User 2	Block	Show			
User 3	Block	Show			

Figure A.20: Overview page of all the users in the system. A list of groups they are connected to can be asked for. Actions to remove, block or edit a certain user are provided. A link to add a new user is not provided: users can only be added to the system when they send a request that can be accepted or denied by the administrator.

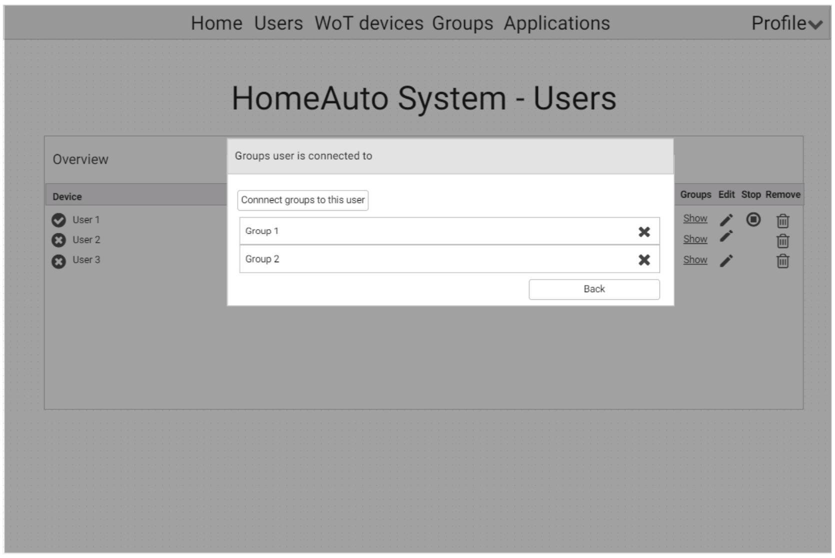


Figure A.21: Showing all the groups a certain user is connected to. Groups can be added or removed.

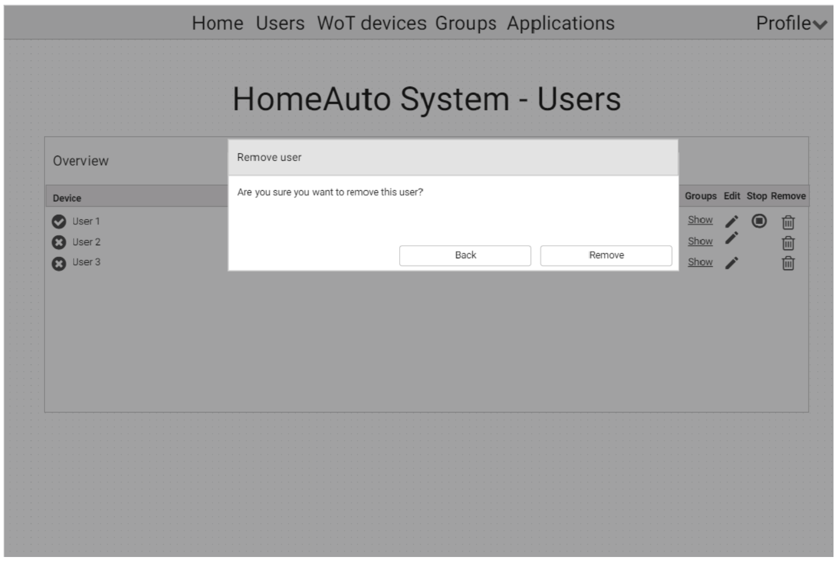
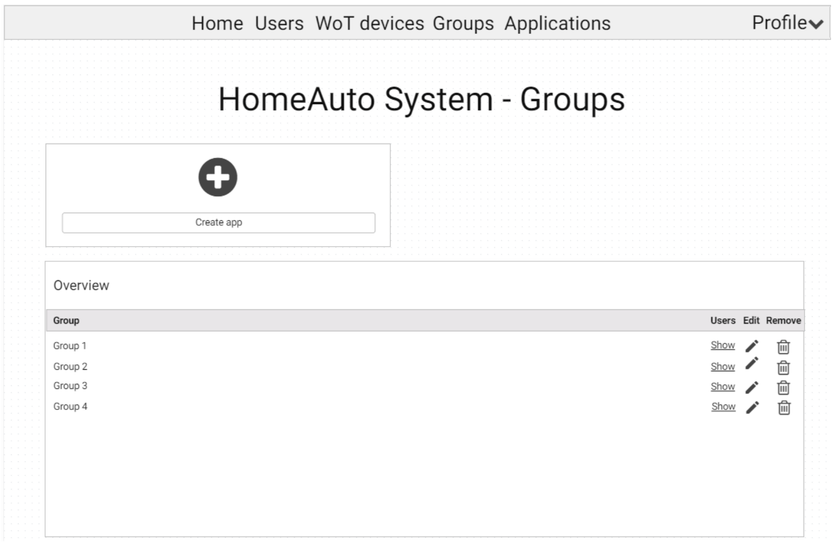


Figure A.22: A confirmation is being asked if the administrator wants to remove a user from the system.



Group	Users	Edit	Remove
Group 1	Show		
Group 2	Show		
Group 3	Show		
Group 4	Show		

Figure A.23: Overview page of all the groups. A list of connected users can be asked for. Actions to edit or remove a certain group are provided.

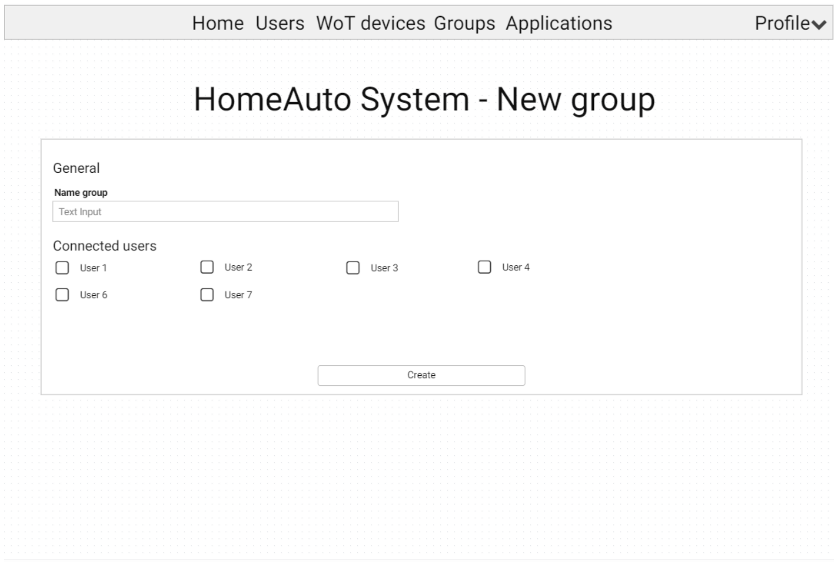


Figure A.24: Creating a new group. A name is required, some users can be connected to it.

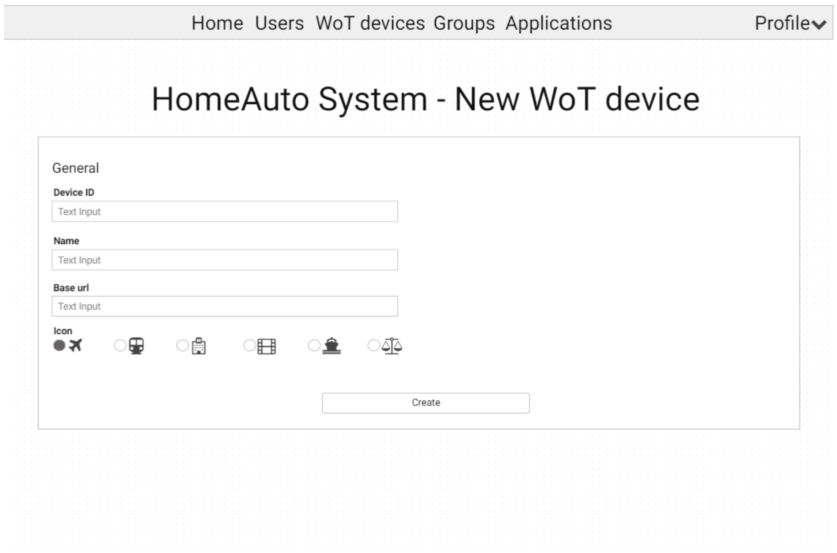


Figure A.25: Adding a new WoT device to the system. An ID has to be passed, together with a name and a base url. After creation, the administrator will automatically be redirected to the page to connect some getters and setters to the device (see next figure).

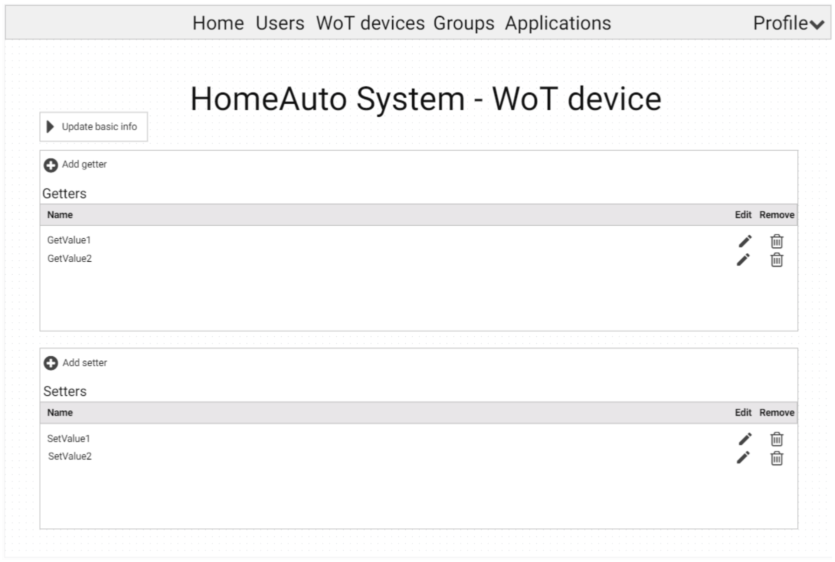


Figure A.26: An overview from all the getters and setters that are already connected to. A link to edit the basic info is provided, as well links to create, update and delete getters and setters.

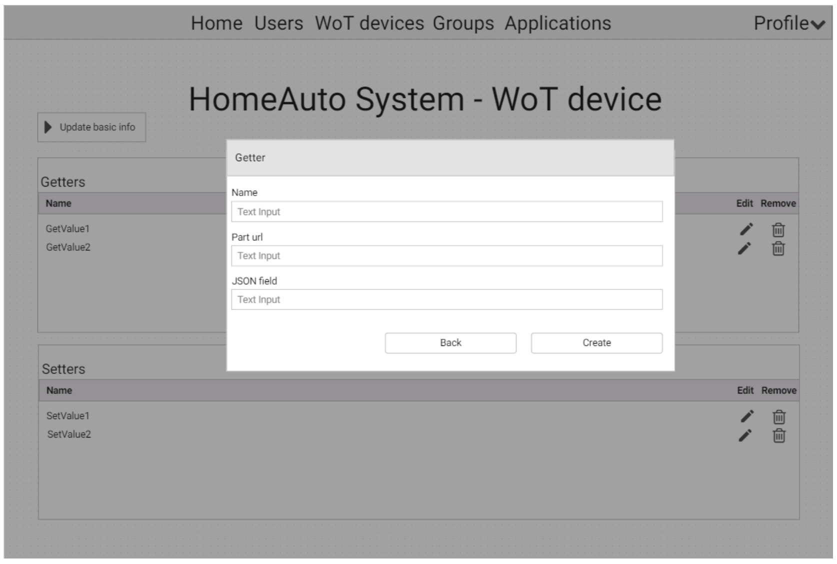


Figure A.27: Adding a getter (or updating one) requires a unique name, a part of a url on which data can be get, and the JSON response field that is used to receive the wanted value.

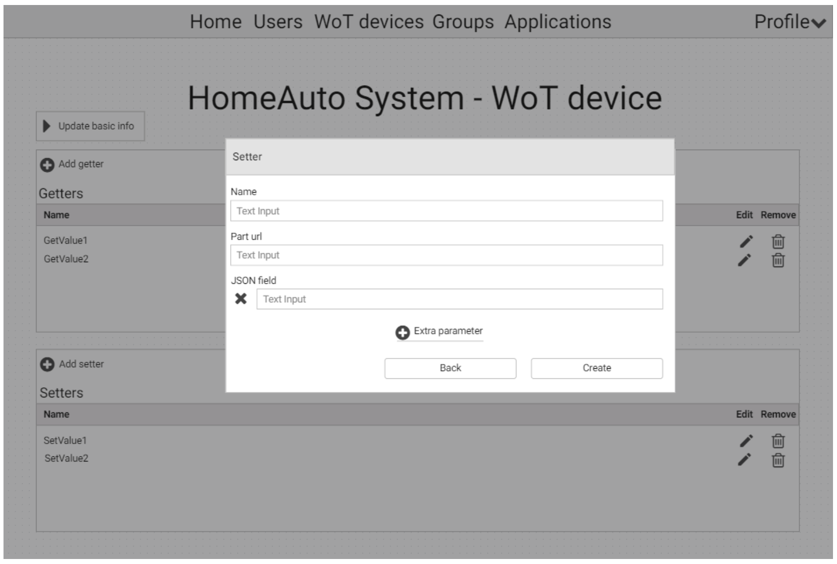


Figure A.28: Adding a getter (or updating one) requires a unique name, a part and zero to many parameters. These parameters will be JSON fields that need to be passed in order to edit some data at the endpoint.

Bibliography

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 54(15):2787–2805, October 2010.

[2] Djamal Benslimane, Schahram Dustdar, and Amit Sheth. Services Mashups: The New Generation of Web Applications. *IEEE Internet Computing*, 12(5):13–15, September 2008.

[3] Erik Berglund and Magnus Bang. Requirements for Distributed User Interface in Ubiquitous Computing Networks. In *Proceedings of Conference on Mobile and Ubiquitous MultiMedia*, Oulo, Finland, December 2002.

[4] Xiaojun Bi and Ravin Balakrishnan. Comparing Usage of a Large High-Resolution Display to Single or Dual Desktop Displays for Daily Work. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1005–1014, Boston, USA, April 2009.

[5] Michael Blackstock and Rodger Lea. IoT Mashups with the WoTKit. In *2012 3rd International Conference on the Internet of Things*, pages 159–166, Wuxi, China, October 2012.

[6] Michael Blackstock and Rodger Lea. Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED). In *Proceedings of the 5th International Workshop on Web of Things*, pages 34–39, Cambridge, USA, October 2014.

[7] Jason Cartwright. Microsoft Shuts Down Popfly. `http://techau.com.au/microsoft-shuts-down-popfly/`, 2009. Retrieved on: 2017-09-17.

[8] Pei-Yu Peggy Chi and Yang Li. Weave: Scripting Cross-Device Wearable Interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3923–3932, Seoul, South Korea, April 2015.

[9] Pei-Yu Peggy Chi, Yang Li, and Bjorn Hartmann. Enhancing Cross-Device Interaction Scripting with Interactive Illustrations. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5482–5493, San Jose, USA, May 2016.

[10] Florian Daniel, Maristella Matera, and Michael Weiss. Next in Mashup Development: User-created Apps on the Web. *IT Professional*, 13(5):22–29, September 2011.

[11] David Dearman and Jeffery S Pierce. It is On My Other Computer!: Computing with Multiple Devices. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 767–776, Florence, Italy, April 2008.

[12] Prasun Dewan and Honghai Shen. Controlling Access in Multiuser Interfaces. *ACM Transactions on Computer-Human Interaction*, 5(1):34–62, March 1998.

[13] Simon Duquennoy, Gilles Grimaud, and Jean-Jacques Vandewalle. The Web of Things: Interconnecting Devices with High Usability and Performance. In *International Conference on Embedded Software and Systems*, pages 323–330, Zhejiang, China, May 2009.

[14] Michael K Edwards. *Making the Best of Chromecast: Amazing Tricks and Tips.* CreateSpace Independent Publishing Platform, February 2015.

[15] Robert J Ennals. Intel Mash Maker: Mashups for the Masses. `https://software.intel.com/en-us/articles/intel-mash-maker-mashups-for-the-masses`, 2012. Retrieved on: 2017-09-17.

[16] Robert J Ennals and Minos N Garofalakis. MashMaker: Mashups for the Masses. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 1116–1118, Beijing, China, June 2007.

[17] Roy T Fielding and Richard N Taylor. *Architectural Styles and the Design of Network-Based Software Architectures.* University of California, January 2000.

[18] Luca Frosini and Fabio Paterno. User Interface Distribution in Multi-Device and Multi-User Environments with Dynamically Migrating Engines. In *Proceedings of the 2014 ACM SIGCHI Symposium on Engi-*

neering Interactive Computing Systems, pages 55–64, Rome, Italy, June 2014.

[19] Jose A Gallud, Ricardo Tesoriero, Jean Vanderdonckt, María Lozano, Victor Penichet, and Federico Botella. *Distributed User Interfaces: Designing Interfaces for the Distributed Ecosystem.* Springer-Verlag, January 2011.

[20] Daniel Giusto, Antonio Iera, Giacomo Morabito, and Luigi Atzori. *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications.* Springer-Verlag, January 2010.

[21] Josh Goldman. Create an API for Any Site with Dapper. `https://techcrunch.com/2006/08/17/create-an-api-for-any-site-with-dapper/`, 2006. Retrieved on: 2017-10-26.

[22] Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. Attach me, Detach me, Assemble me Like you Work. In *roceedings of the 2005 IFIP TC13 International Conference on Human-Computer Interaction*, volume 5, pages 198–212, Rome, Italy, September 2005.

[23] William I Grosky, Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An Infrastructure for Shared Sensing. *IEEE multimedia*, 14(4):8–13, January 2007.

[24] Dominique Guinard and Vlad Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW*, Madrid, Spain, April 2009.

[25] Dominique Guinard and Vlad Trifa. *Building the Web of Things: With Examples in Node. js and Raspberry Pi.* Manning Publications Co., June 2016.

[26] Dominique Guinard, Vlad Trifa, Thomas Pham, and Olivier Liechti. Towards Physical Mashups in the Web of Things. In *Proceedings of the 6th International Conference on Networked Sensing Systems*, pages 196–199, Pittsburgh, USA, June 2009.

[27] Dominique Guinard, Vlad Trifa, and Erik Wilde. A Resource Oriented Architecture for the Web of Things. In *Internet of Things (IOT)*, pages 1–8, Tokyo, Japan, November 2010.

[28] Bjorn Hartmann, Leslie Wu, Kevin Collins, and Scott R Klemmer. Programming By a Sample: Rapidly Creating Web Applications with D. Mix. In *Proceedings of The 20th Annual ACM Symposium On User interface Software and Technology*, pages 241–250, Newport, USA, October 2007.

[29] Tommi Heikkinen, Jorge Goncalves, Vassilis Kostakos, Ivan Elhart, and Timo Ojala. Tandem Browsing Toolkit: Distributed Multi-Display Interfaces with Web Technologies. In *Proceedings of The International Symposium on Pervasive Displays*, pages 142–147, Copenhagen, Denmark, June 2014.

[30] Steven Houben and Nicolai Marquardt. Watchconnect: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1247–1256, Seoul, South Korea, April 2015.

[31] Maria Husmann, Michael Nebeling, and Moira C Norrie. MultiMasher: a Visual Tool for Multi-Device Mashups. In *International Conference on Web Engineering*, pages 27–38, Aalborg, Denmark, July 2013.

[32] Dugald Ralph Hutchings, Greg Smith, Brian Meyers, Mary Czerwinski, and George Robertson. Display Space Usage and Window Management Operation Comparisons Between Single Monitor and Multiple Monitor Users. In *Proceedings of the Working Conference on Advanced visual interfaces*, pages 32–39, Gallipoli, Italy, May 2004.

[33] Andreas Kamilaris, Andreas Pitsillides, and Vlad Trifa. The Smart Home Meets the Web of Things. *International Journal of Ad Hoc and Ubiquitous Computing*, 7(3):145–154, May 2011.

[34] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, et al. People, Places, Things: Web Presence for the Real World. *Mobile Networks and Applications*, 7(5):365–376, December 2002.

[35] Robert Kleinfeld, Stephan Steglich, Lukasz Radziwonowicz, and Charalampos Doukas. glue. things: a Mashup Platform for Wiring the Internet of Things with the Internet of Services. In *Proceedings of the 5th International Workshop on Web of Things*, pages 16–21, Cambridge, USA, October 2014.

[36] Agnes Koschmider, Victoria Torres, and Vicente Pelechano. Elucidating The Mashup Hype: Definition, Challenges, Methodical Guide and

Tools for Mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW*, pages 1–9, Madrid, Spain, April 2009.

[37] Thomas Kubitza, Alexandra Voit, Dominik Weber, and Albrecht Schmidt. An IoT Infrastructure for Ubiquitous Notifications in Intelligent Living Environments. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, pages 1536–1541, Heidelberg, Germany, September 2016.

[38] Danh Le Phuoc, Axel Polleres, Giovanni Tummarello, and Christian Morbidoni. DERI Pipes: Visual Tool for Wiring Web Data Sources. *Book DERI pipes: Visual Tool for Wiring Web Data Sources*, January 2008.

[39] James R Lewis. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. *International Journal of Human-Computer Interaction*, 7(1):57–78, January 1995.

[40] Tony Loton. *Introduction to Microsoft Popfly, No Programming Required.* Lotontech Limited, February 2008.

[41] Kris Luyten and Karin Coninx. Distributed User Interface Elements to Support Smart Interaction Spaces. In *Seventh IEEE International Symposium on Multimedia*, pages 277–286, California, USA, December 2005.

[42] Luca Mainetti, Vincenzo Mighali, Luigi Patrono, Piercosimo Rametta, and Silvio Lucio Oliva. A Novel Architecture Enabling the Visual Implementation of Web of Things Applications. In *2013 21st International Conference on Software, Telecommunications and Computer Networks*, pages 1–7, Primosten, Croatia, September 2013.

[43] Jose Pascual Molina Masso, Jean Vanderdonckt, Pascual Gonzalez Lopez, Antonio Fernandez-Caballero, and Maria Dolores Lozano Perez. Rapid Prototyping of Distributed User Interfaces. In *Proceedings of 6th International Conference on Computer-Aided Design of User Interfaces CADUI 2006*, pages 151–166, Bucharest, Romania, June 2006.

[44] Jeremie Melchior, Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. A Toolkit for Peer-to-Peer Distributed User Interfaces: Concepts, Implementation, and Applications. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 69–78, Pittsburgh, USA, July 2009.

[45] Jeremie Melchior, Jean Vanderdonckt, and Peter Van Roy. A Model-Based Approach for Distributed User Interfaces. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 11–20, Pisa, Italy, June 2011.

[46] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of Things: Vision, Applications and Research Challenges. *Ad Hoc Networks*, 10(7):1497–1516, September 2012.

[47] Alan Moore and David Redmond-Pyle. *Graphical User Interface Design and Evaluation: A Practical Process.* Prentice Hall, May 1995.

[48] Sudarshan Murthy, David Maier, and Lois Delcambre. Mash-o-Matic. In *Proceedings of the 2006 ACM Symposium on Document Engineering*, pages 205–214, Amsterdam, The Netherlands, October 2006.

[49] Daniel Nations. What Does Web 2.0 Even Mean? `https://www.lifewire.com/what-is-web-2-0-p2-3486624`, 2016. Retrieved on: 2017-09-17.

[50] Michael Nebeling. xdbrowser 2.0: Semi-automatic generation of cross-device interfaces. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4574–4584, Denver, USA, May 2017.

[51] Michael Nebeling and Anind K Dey. XDBrowser: User-Defined Cross-Device Web Page Designs. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5494–5505, San Jose, USA, May 2016.

[52] Michael Nebeling, Theano Mintsi, Maria Husmann, and Moira Norrie. Interactive Development of Cross-Device User Interfaces. In *Proceedings of the 32nd Annual ACM Conference on Human factors in Computing Systems*, pages 2793–2802, Toronto, Canada, April 2014.

[53] Dan O'Sullivan and Tom Igoe. *Physical Computing: Sensing and Controlling the Physical World with Computers.* Course Technology Press, May 2004.

[54] Ahmed Patel, Liu Na, Rodziah Latih, Christopher Wills, Zarina Shukur, and Rabia Mulla. A Study of Mashup as a Software Application Development Technique with Examples from an End-User Programming Perspective. *Journal of Computer Science*, 6(12):1406–1415, November 2010.

[55] Cesare Pautasso and Erik Wilde. Why Is the Web Loosely Coupled: a Multi-Faceted Metric for Service Design. In *Proceedings of the 18th International Conference on World Wide Web*, pages 911–920, Madrid, Spain, April 2009.

[56] Rajeev Piyare, Sun Park, Se Yeong Maeng, Sang Hyeok Park, Seung Chan Oh, Sang Gil Choi, Ho Su Choi, and Seong Ro Lee. Integrating Wireless Sensor Network into Cloud Services for Real-Time Data Collection. In *2013 International Conference on ICT Convergence*, pages 752–756, Jeju, South Korea, October 2013.

[57] Michael E Porter and James E Heppelmann. How Smart, Connected Products are Transforming Competition. *Harvard Business Review*, 92(11):64–88, November 2014.

[58] Mark Pruett. *Yahoo! pipes*. O'Reilly, May 2007.

[59] Reza Rawassizadeh, Blaine A Price, and Marian Petre. Wearables: Has the Age of Smartwatches Finally Arrived? *Communications of the ACM*, 58(1):45–47, January 2015.

[60] Claire Rowland, Elizabeth Goodman, Martin Charlier, Ann Light, and Alfred Lui. *Designing Connected Products: UX for the Consumer Internet of Things*. O'Reilly Media, May 2015.

[61] David E Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh. Damia: Data Mashups for Intranet Applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1171–1182, Vancouver, Canada, June 2008.

[62] Ryan Singel. Map Hack on Crack. `https://www.wired.com/2005/07/map-hacks-on-crack/`, 2005. Retrieved on: 2017-09-17.

[63] Micael Sjolund, Anders Larsson, and Erik Berglund. Smartphone Views: Building Multi-Device Distributed User Interfaces. *Mobile Human-Computer Interaction MobileHCI 2004. Lecture Notes in Computer Science*, 3160(1):127–140, September 2004.

[64] Nate Swanner. Yahoo Is Shutting Down Maps, Pipes and a Bunch of Services You've Probably Never Heard of. `https://thenextweb.com/insider/2015/06/04/yahoo-is-shutting-down-maps-pipes-and-a-bunch-of-services-youve-probably-never-heard-of/`, 2015. Retrieved on: 2017-09-17.

[65] Desney S Tan, Brian Meyers, and Mary Czerwinski. WinCuts: Manipulating Arbitrary Window Regions for More Effective Use of Screen Space. In *CHI 2004 Extended Abstracts on Human Factors in Computing Systems*, pages 1525–1528, Vienna, Austria, April 2004.

[66] Lucia Terrenghi, Aaron Quigley, and Alan Dix. A Taxonomy for and Analysis of Multi-Person-Display Ecosystems. *Personal and Ubiquitous Computing*, 13(8):583–598, November 2009.

[67] Stefan Thomke and Eric Von Hippel. Customers as Innovators: a New Way to Create Value. *Harvard Business Review*, 80(4):74–85, April 2002.

[68] Rob Van der Meulen. Gartner Says 8.4 Billion Connected Things Will Be in Use in 2017, Up 31 Percent From 2016. `https://www.gartner.com/newsroom/id/3598917`, 2017. Retrieved on: 2017-09-17.

[69] Peter Van-Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. MIT press, Cambridge, USA, 2004.

[70] CP Vandana, Suman Thapa, and Pradip Thapa. Web of Things. *International Journal of Scientific Research In Computer Science, Engineering and Information Technology*, 2(3):212–218, May 2017.

[71] Jean Vanderdonckt et al. Distributed User Interfaces: How To Distribute User Interface Elements Across Users, Platforms, and Environments. In *Proceedings of XIth Congreso Internacional de Interaccion Persona-Ordenador Interaccion 2010*, pages 3–14, Valencia, Spain, September 2010.

[72] Roy Want, Kenneth P Fishkin, Anuj Gujar, and Beverly L Harrison. Bridging Physical and Virtual Worlds with Electronic Tags. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 370–377, Pittsburgh, USA, May 1999.

[73] Erik Wilde. Putting things to REST. November 2007.

[74] Jeffrey Wong and Jason I Hong. Making Mashups with Marmite: Towards End-User Programming for the Web. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1435–1444, San Jose, USA, April 2007.

[75] Felix Wortmann and Kristina Fluchter. Internet of Things. *Business & Information Systems Engineering*, 57(3):221–224, June 2015.

[76] Jishuo Yang and Daniel Wigdor. Panelrama: Enabling Easy Specification of Cross-Device Web Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2783–2792, Toronto, Canada, April 2014.

[77] Youngjin Yoo, Ola Henfridsson, and Kalle Lyytinen. Research Commentary âĂŤ The New Organizing Logic of Digital Innovation: An Agenda for Information Systems Research. *Information Systems Research*, 21(4):724–735, December 2010.

[78] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding Mashup Development. *IEEE Internet computing*, 12(5), September 2008.

[79] Nan Zang, Mary Beth Rosson, and Vincent Nasser. Mashups: Who? What? Why? In *CHI 2008 Extended Abstracts on Human Factors in Computing Systems*, pages 3171–3176, Florence, Italy, April 2008.