



Universität Stuttgart

Institute of Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Master's Thesis Nr. 3118-0008

Exploration of Programming by Demonstration Approaches for Smart Environments

Ragavendra Lingamaneni

Course of Study: INFOTECH

Examiner: Prof. Dr. Albrecht Schmidt

Supervisor: Thomas Kubitza
M.Sc

Commenced: 2015-09-07

Completed: 2016-03-02

CR-Classification: H.5.2, I.2.7

Acknowledgements

I would like to gratefully acknowledge the guidance, help and motivation of my supervisor Thomas Kubitza and with immense gratitude that I would like to thank Prof. Dr. Albrecht Schmidt for giving me an opportunity to do my thesis at the HCI department.

Special thanks to my friends Sujata Roy Chowdhury and Sujith Gowda for all their support and motivation. I am really thankful to them for creating an enjoyable work environment throughout the entire duration of this thesis.

Abstract

The number of smart electronic devices like smartphones, tablet computers and embedded sensors/actuators in our domestic and work environment is constantly growing. Some of them work as a stand along devices while others already collaborate with each other. It is apparent that once a common layer for device intercommunication between major consumer device manufactures has been agreed upon, a new class of networked smart applications will rise. These applications will dynamically utilise required sensors and actuators of a smart environment to optimally achieve tasks for us human users. Inhabitants of such environments are already interacting with dozens of computers per day. A lot of research has addressed many issues in hardware and software for the future smart environments But few have focused on the users. An important research topic lies in finding simple, intuitive yet powerful enough approaches to allow end-users to create and modify the behaviour of smart environments in which they live and work according to their needs. I believe that for the ubiquitous computing environments to reach its full potential, enabling end-user programming is one of the important criteria.

This thesis describes the exploration of various approaches for "Do It Yourself" philosophy in smart environment applications by providing inhabitants with the appropriate tools which empower them to build their environments in accordance to their needs and with enough room for personal creativity. To this end, I choose speech as the main input by the end users along with demonstration of certain parts of over all approach in building applications for smart environments. The resulting application is built on top of the meSchup platform developed during meSchup FP7 EU project at the VIS institute in Stuttgart which provides a middle-ware for seamlessly interconnecting heterogeneous devices. The resulting web application is called "Speechweaver" which combines speech, programming by demonstration and automatic code generation into usable and intuitive approach for creating and modifying the rule based behaviour of smart environments in place.

Contents

Abstract	iii
1 Introduction	1
1.1 Overview	1
1.2 Motivation	1
1.3 Problem Statement	2
1.4 Thesis Outline	2
2 Background and Related Work	5
2.1 Ubiquitous Computing	5
2.2 End-User Programming	6
2.3 End-User Programming for Ubiquitous Computing	6
2.3.1 Visual Programming	7
2.3.2 Form based Programming	8
2.3.3 Programming by Example	8
2.3.4 Tangible Programming	9
2.3.5 Programming with Simplified Natural Language	10
2.4 Human-Computer Interaction through Speech Interfaces	11
3 System Concept	13
3.1 Interaction Goals	13
3.2 Interaction Evolution	14
3.3 Interaction Technique	14
3.3.1 Multimodal Systems	14
3.3.2 Mobile vs Desktop vs non GUI Applications	15
3.3.3 Web application vs Native application	15
3.4 End-user Application Development Analogies	16
3.5 Choosing Abstractions	17
3.5.1 Ready Made Templates	17
3.5.2 User Defined Templates	18
3.6 Collecting Ingredients (Device Referencing)	18
3.6.1 Searching among all Connected Devices List	18
3.6.2 Device Referencing by Location	19
3.6.3 Device Referencing by Physical Proximity	19
3.6.4 Device Referencing by Direct Physical Manipulation	21
3.7 Content Elements	21
3.8 Live Value Element	21
3.9 User Interface Design	22

3.9.1	Behaviour editor design	23
3.9.2	Cooking Panel (Speech Area) Design	24
3.9.3	Component element design	24
3.10	Speechweaver Spoken Sentence Structures	25
3.11	Application Development in Speechweaver	26
4	Implementation	29
4.1	System Overview	29
4.2	meSchup Middleware	30
4.2.1	Rule Engine	30
4.2.2	Device Ontologies	31
4.2.3	REST API	31
4.3	Technologies used for Application Development	31
4.3.1	Web Components	31
4.3.2	Polymer.js	32
4.4	Speechweaver Application Architecture	32
4.4.1	Important Application Elements	33
4.5	How does Speechweaver work?	35
4.5.1	Connecting to meSchup Server	35
4.5.2	Initializing the Application	36
4.5.3	Recipe and Behaviour Creation	36
5	Evaluation	39
5.1	Evaluation Objectives	39
5.2	Evaluation Methodology	39
5.3	Smart Environment Test Setup	40
5.4	Participant Tasks	42
5.5	Results	43
5.6	Discussion	47
6	Conclusion and Future Work	51
6.1	Summary	51
6.2	Discussion	51
6.3	Future Work	53
A	User Study	55
A.1	Pre-Study Questionnaire	56
A.2	Questionnaire for Tasks	59
A.3	Post Study Questionnaire	62
A.4	System Usability Study Questionnaire	65
	Bibliography	67

List of Figures

2.1	Visual programming examples	7
2.2	A Sample IFTTT Recipe	8
2.3	The ToonTalk programming environment	9
2.4	Tangible programming examples	10
2.5	CAMP interface	11
2.6	Amazon Echo	11
3.1	A ready made template ingredient	18
3.2	All the connected device list in a test smart environemnt	19
3.3	A list of all the locations to which devices are tagged	19
3.4	Approaches for referencing devices by proximity	20
3.5	Different content elements	21
3.6	Live Value Token	22
3.7	Live value token in manual edit mode	22
3.8	Behaviour Editor Screen	23
3.9	Speech area	24
3.10	Sensor component ingredient design	24
3.11	A module representation along with it capabilities	25
3.12	Sample Recipe list view	26
3.13	Sample Behaviour list view	27
4.1	System Overview	29
4.2	meSchup Platform	30
4.3	Speechweaver Application Architecture	32
4.4	Visual feedback of annotated spoken behaviour	33
4.5	Application settings panel	36
5.1	Test Smart Environment	41
5.2	Task 1 Smart Objects	42
5.3	Task 2 Smart Objects	43
5.4	Participants rating for the statement "System had understood me well" after T1	43
5.5	Participants rating for the statement "System had understood me well" after T2	44
5.6	Participants rating for the statement "System had understood me well" after T3	45
5.7	Participants SUS score	45
5.8	Participants rating for the statement "I found it helpful that the system works on a mobile device"	46
5.9	Participants rating for the statement "I find the “ingredients” concept easy to understand"	46

List of Figures

5.10	Participants rating for the statement "I found adding devices by manipulation helpful"	47
5.11	Participants rating for the statement "I found the visual feedback while speaking helpful"	47

Chapter 1

Introduction

1.1 Overview

Smart environments are already fundamentally changing the way we live and interact with the environment. With the envisioned smart homes, smart workplaces, smart class rooms no human space will be left “dumb”. A common example of the future of smart environments is the smart home. In a smart home, sensors are embedded in the walls, ceilings, floors, and appliances inside a private residence. These sensors monitor a diverse set of properties from the environment and readings are then used to draw high level conclusions about the state of the home. Based on these conclusions and some set of configured behaviors, a smart home can engage actuators to affect changes in the environment. Typical smart home behaviors are designed to make the home’s occupants more comfortable and more productive.

As the trend towards technology-enriched smart environments increases, the need to enable end users to create applications according to their needs arise. There are several recent research projects which are focusing on lowering application creation barriers by using simplified programming languages and input mechanisms. However, very few have explored the combination of speech and demonstration for interacting with the devices around them and program them according to their needs.

This thesis introduce Speechweaver, a multimodal browser based web application with an interaction approach that shows a way for end-users to configure their smart environments. Speechweaver discovers available ubiquitous components and presents these components as live widgets which are used when describing behaviour by speech and in some parts by demonstration. It is built on top of an a existing infrastructure in the form of middleware called meSchup developed to support writing scripts for recombination of devices and their services. This thesis also present the system evaluation and challenges inherent in such a system.

1.2 Motivation

The technology necessary to create smart environments like sensors, actuators, computers, wireless communication networks, etc are already available. However, what is missing is how to build unique applications specific to each smart environment. Completely general purpose

solutions are infeasible due to wide range of devices and their services that are available that makes every smart environment unique.

The motivation behind Speechweaver was the belief that for ubiquitous computing to reach its full potential, providing end-user programming is the most important criteria. Computing power is already everywhere around us, in cars, refrigerators, TVs, mobile phones, tablets and many more devices. Missing is the software to glue these devices and make them work for us.

The author of this thesis believe that end-user programming capabilities are an essential part of any flexible ubiquitous computing environments. In such an environment users wish to configure, connect and program in such a way that no application developer has foreseen. But we can expect users to have knowledge about the devices they encounter in their environments. It is the user who understands what what he can do with a particular service provided by a device. This enables users to benefit fully from the possibilities ubiquitous computing offers.

There is also a possibility that users will come up with previously unknown relations between the sensor triggers like, door opening is detected by pressure sensor. This will enable playfulness with the objects around where ideas are generated from exploration and observation.

1.3 Problem Statement

The goal of the thesis is to let end-users create own applications for their smart environments according to their needs without relying on technicians or other programmers. While there is a significant research related to end-user programming, user interacting with physical things need to have new interaction techniques. This requires us to come up with a user interaction concept that empowers the inhabitants of a smart environment to program cross-device behaviour especially without the need to have expertise in programming languages or a technical background.

The work presented in this thesis combines the idea of ubiquitous computing with the possibilities of programming by speech and in some parts by demonstration. The goal is not to allow the user to use completely unconstrained natural language for any possible automation task. Speech input will be structured with templates (eg. if-this-then-that) but still feels natural for the users to speak. However, the system can be extended to accommodate other templates by adding extra information. Users must be able to wire together the available device capabilities into applications that meet their needs.

1.4 Thesis Outline

The remainder of this thesis is organised as follows.

- **Chapter 2** : This chapter looks into the background and investigates related work in the area of end-user programming of smart environments.

- **Chapter 3** : This chapter details the evolution of interaction concept of the Speech-weaver application.
- **Chapter 3** : This chapter provides the implementation details of the system.
- **Chapter 4** : This chapter discusses about evaluation and its results.
- **Chapter 5** : The final chapter provides a summary of this work, limitations and discusses directions for future work.

Chapter 2

Background and Related Work

This chapter will discuss about general background in ubiquitous computing and smart environments as well as end-user programming in ubiquitous computing environments and its related world.

2.1 Ubiquitous Computing

Ubiquitous computing [23] or pervasive computing is a paradigm where computing disappear into the background of everyday life and is unobtrusive. It is a vision where computers will be integrated seamlessly into the environment like in couches, walls, cars, clothing etc to support everyday interaction with these objects.

Implementation of ubiquitous computing relies on the convergence of wireless technologies, advanced electronics and the Internet. Sensors play a major role in ubiquitous computing as timely and accurate picture of the surrounding environment in which activity is occurring is very important for such a system.

Ubiquitous computing allows the user to focus on the information required to make smart decisions rather than dealing with the technology itself. Hence representation of information obtained from ubiquitous devices is more important. Ubiquitous computing also brings in problems like how to find or identify all these embedded computing devices and sensors to access their services.

Smart Environments

Smart environments are envisioned as the byproduct of ubiquitous computing and the availability of cheap computing power that enables the development of smart devices which are continuously working to make inhabitant's lives more comfortable and productive.

Mark Weiser [23] described “smart environment” as “a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network”.

The authors view of smart environments can be summarized as listed below :

- A physical world interwoven with invisible sensors, actuators, displays, and computational elements.
- The smartness of this environment is a product of interaction of different devices and computing systems.
- An ecosystem of interacting objects, e.g. sensors, devices, appliances and embedded systems in general, that have the capability to provide rich end-user services.

2.2 End-User Programming

End-user programming (EUP) or end-user development (EUD) refers to approaches that help to make programming more accessible to a large group of people which includes people with no previous programming skills.

While only a few computer users actually know how to program in usual programming languages, almost all users want to modify existing applications by adding additional features or specifying their behavior, and even inventing new ones becomes important when looking at technology-enriched environments like smart homes, offices etc. There would be a greater benefit of the added technology when users were able to decide how their devices should work and react instead of just using pre-defined actions [10]

End-user programming systems try to make the abstract and high-level concepts that are required to program computers understandable for non-expert users. These systems lower the entry barrier for users (low threshold), but also try to provide powerful and flexible functionality (high ceiling) to create new envisioned applications.

These end-user programming environments typically fall into two large groups [12]: Systems that teach people how to program and systems that empower people to build things that are tailored towards their needs, which are of particular interest to us.

2.3 End-User Programming for Ubiquitous Computing

Weiser claimed that the whole point of ubiquitous computing was to create compelling applications that would drive the development of devices and infrastructure. Machine learning can eliminate the need for humans in the loop by automatically learning an inhabitant's routine. However, users will lose control and the ability to extend their environment.

As ubiquitous computing matures, and becomes a part of everyday life, the way in which users control it will become increasingly important. While direct or implicit control may suffice for many applications, the sheer diversity of situations in which ubiquitous technology may be used means that users will, at some point, want to tailor the technology to their own specific needs.

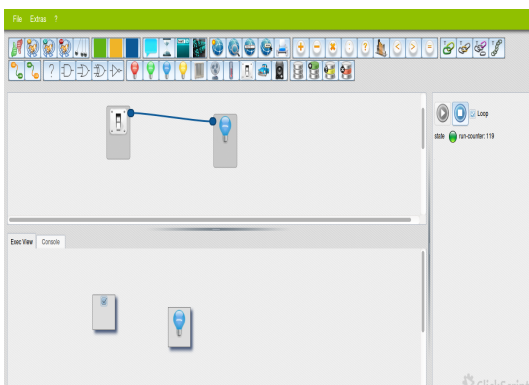
The system must allow users to access devices and their services are available in their environment . This implies that devices and services must be able to provide enough information about themselves in human readable form, so that users can make informed decisions about wiring them together.

Some of the various approaches taken for end-user programming for ubiquitous computing along with their related work is discussed in the following sections.

2.3.1 Visual Programming

On widespread approach in visual programming is to wire abstract concepts by linking inputs and outputs of specific widgets in a graphical user interface.

These systems provide graphical visualizations of the program functionality. Usually these are visualizations of fundamental building blocks that can be connected by the end-user to construct the desired program behavior. These visualizations can become very complex and confusing for more sophisticated programs.



(a) ClickScript Editor



(b) Reality Editor

Figure 2.1: Visual programming examples

ClickScript

Clickscript [6] is a web mashup editor and a Firefox plug-in that allows visual creation of Web mashups by connecting building blocks of resources (websites) and operations (greater than, if/then, loops, etc.). The visual languages are used in combination with property dialogs for the parameterization of mashup elements. Data sources are primarily accessed through mashup elements in those notations. Dialog-based wiring of widgets provides an alternative to visual data-flow languages which saves screen space, but depends on visual widgets. On an intermediate level of abstraction, there is often a distinction between design and run-time mode, although the mashup development environment try to reduce this distinction, e.g., by using previews.

Reality Editor

Reality Editor [7] is a system that supports editing the behavior and interfaces of smart objects using augmented reality technology. Reality Editor augments graphical elements directly on top of the tangible interfaces found on physical objects such as push buttons or knobs which are tagged with markers. It allows reprogramming of the interfaces and behavior of the objects as well as defining relationships between smarter objects in order to easily create new functionalities.

2.3.2 Form based Programming

In form based end user programming, users are required to fill forms or predefined templates by choosing relevant options provided to fulfil their tasks.

IFTTT

IFTTT is a web tool that allows users to create simple programs with "triggers" and "actions". For example, one can program their Phillips Hue light bulbs to flash red and blue when the Cubs hit a home run. A GUI allows users to construct these recipes based on a set of information "channels". These channels represent many types of information. Weather, news, and financial services have provided constant updates through web services. Home automation sensors and controllers such as motion detectors, thermostats, location sensors, garage door openers, etc. are also available. Users can create their own recipe and then describe the recipes they have constructed in natural language and publish them.



Figure 2.2: A Sample IFTTT Recipe

2.3.3 Programming by Example

In programming by example approach, the user records desired automation by manually performing the task to be automated and turns it into a program by using a direct manipulation interface. With this approach, users can directly demonstrate parts of the program behavior and the end-user programming system build the necessary application. Macro recorder is one

of the most straight forward programming by example approach where user simply recorded a sequence of actions and repeat them later at some point in time by giving appropriate commands.



Figure 2.3: The ToonTalk programming environment

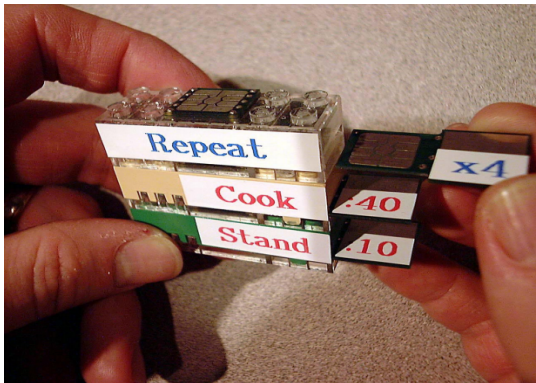
Another example in this approach is ToonTalk [11] which is aimed at school going children, in which the source code is animated and the programming environment is a video game. Since the target audience are children every abstract computational aspect is mapped into a concrete metaphor. For example, a computation is a city, an active object or agent is a house, birds carry messages between houses, a method or clause is a robot trained by the user and so on. The programmer trains a 'programmer persona' a robot that can learn by watching user actions. To train the robot, user performs the intended actions on some input. When the robot subsequently sees the same input, it will perform the same actions that the user did.

2.3.4 Tangible Programming

In tangible programming approach, programming structures take physical forms. For example, a "method" or a "if" statement in a program will be a physical object which can be passed around in a room.

Tangible Programming Bricks

Tangible programming bricks [15] embeds electronics on the physical objects that actually perform the commands themselves. These bricks can be stacked upon other bricks in upward direction to physically build a program. The connector system used for stacking bricks relies on the plastic knobs and tubes of the LEGO SYSTEM for physical clutching [14]. Additionally it employed "inset card" system for providing various inputs to the bricks.



(a) MIT's Tangible Programming Bricks



(b) Quetzal - Programming parts

Figure 2.4: Tangible programming examples

Quetzal

In Quetzal [8] programming language, users have to physically link the tangible programming objects. Tangible objects embody the meaning of the instructions in a programming language. Quetzal is developed for teaching programming for middle school and late elementary school children for controlling LEGO Mindstorms robots.

2.3.5 Programming with Simplified Natural Language

With this approach, the end-user programming systems try to infer the desired program directly from the users' instructions. These systems are often implemented as a dialog between the computer and the user, and sometimes combined with the programming by demonstration technique. However, it is very complex for users to modify existing programs afterwards; therefore, the systems are sometimes provide a graphical feedback similar to the visual programming systems.

CAMP: a magnetic poetry interface

CAMP [19] is a programming system which enables end-users to build ubicomp applications for their home. The actions supported by CAMP are capture, access, and delete, and the possible capture data types are still-pictures, audio, and video. CAMP requires that the used technology is context-aware, i.e. it can recognize the contexts of people and objects, such as locations and activities. The CAMP interface consists of a vocabulary area, a poem authoring area, and an interpretation area. To form poems, the user can drag magnets from the vocabulary to the authoring area. When pressing the "run" button, CAMP displays an interpretation.

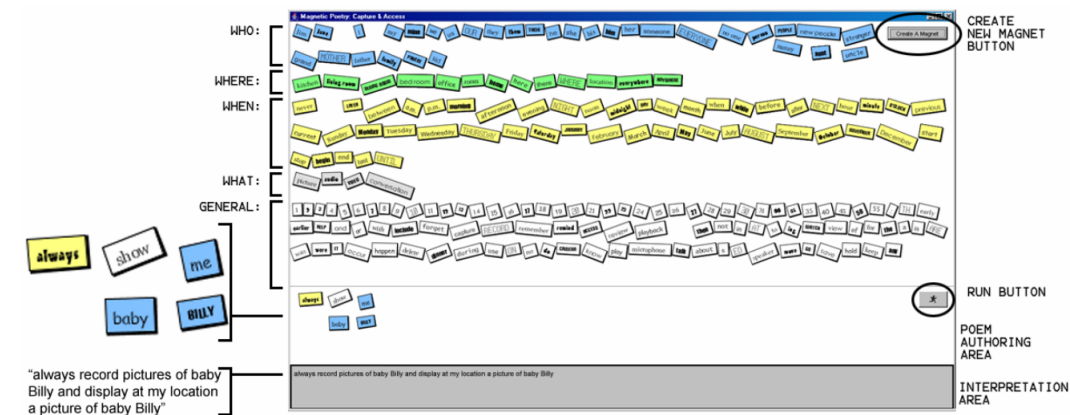


Figure 2.5: CAMP interface

2.4 Human-Computer Interaction through Speech Interfaces

Speech interfaces play a crucial role in enhancing human-computer interactions especially in smaller mobile devices like smart phones for which graphical user interface has obvious limitations considering that user input on mobile devices is rather inconvenient due to the lack of a full hardware keyboard, it becomes obvious that such devices are a good platform for voice-controlled applications utilizing cloud-based speech recognition.

For programming by speech only approaches in smart environments, dialogue based systems promises best solution as interactive dialogue can be used to seek clarifications and avoid misunderstandings and to provide rich feedback.



Figure 2.6: Amazon Echo

There are several spoken dialogue systems in the mobile application industry. Two of the most popular applications are Apple's personal assistant Siri and Google's personal assistant Google Now. Another widely known application featuring highly sophisticated natural language and speech processing technology is the question answering system IBM Watson famous for beating the two ultimate champions in the quiz show Jeopardy. Another recent addition to speech

recognition systems used in home environment is Amazon Echo. It is a wireless speaker and voice command device and is capable of voice interaction, music playback, making to-do lists, setting alarms, and providing weather, traffic and other real time information. It can also control several smart devices.

Chapter 3

System Concept

This chapter will discuss about the evolution of speechweaver interaction techniques and approaches along with other possible alternatives.

3.1 Interaction Goals

In a publication by Gregory D. Abowd[1], he suggests shifting the focus of Ubiquitous Computing research towards design problems as the necessary technologies for creating smart environments already exists but lacks good interaction techniques for the people to interact and manipulate and program these smart environments.

Taking motivation from the above suggestion, this section of thesis investigates various interaction techniques and approaches that enable the inhabitants of the smart environments to create applications in the form of cross-device behaviours without necessarily requiring programming expertise, just by using natural interactions like observation, physical manipulation, speech and rational thinking.

One of the starting point of this work is the belief that the developed application should support the ability of users to intermix services provided by the devices surrounding them. Additionally, the following design goals motivated the development of Speechweaver interaction approach :

- Enable end-users to express their ideas in the same way they think about them for creating smart environment behaviours [16]
- Provide a tool that is live & reactive, which enables end-users to create, visualize, test and debug behaviours in their environment without relying on advanced developers.
- Provide different levels of abstraction to get better understanding over various devices and their capabilities present in their environment.
- Provide adequate User-Interface (UI), right tools and methods to find smart devices in their environment.

3.2 Interaction Evolution

Some of the important user interaction with Speechweaver application involves creating new behaviour, modifying existing behaviour and removing a behaviour from their smart environment. The author believes that there is a common general approach that an end-user will take for doing any of the above mentioned tasks. For example, the process of interaction evolution with Speechweaver application by a user for modifying an existing behaviour consists of the following stages :

- **Identify opportunities** : Inhabitant of a smart environment identifies an opportunity to change the behaviour of the environment for example, he observes that lights are turning on even when the room is sufficiently bright.
- **observe changes** : Inhabitant observes the environment carefully to determine what is triggering the change and when the change happens.
- **Decision making** : Inhabitant decides on if the environment behaviour needs to be changed and if so, how it should be done and by what measure.
- **Describe behaviour to system** : Inhabitant describes the changes to the environment and checks immediately if it behaves how he intended it to behave.

The whole process occur iteratively until the inhabitant is satisfied with the behaviour. For creating a new behaviour the above stages holds true with one extra step to check for the availability of devices for fulfilling the identified opportunities.

3.3 Interaction Technique

This section will discuss about the various decisions that were made to come up with a final feasible interaction techniques and approaches from various alternatives for the Speechweaver.

3.3.1 Multimodal Systems

Humans interact with the physical world multimodally. We employ multiple senses to explore, experience and understand the surrounding environment. Hence there is a need to develop multimodal interfaces for the user interactions in a smart environment. Using only traditional WIMP schema (Windows, Icons, Menus and Point Device) based interaction between humans and computation will be less effective and in some cases not feasible. Multimodal interfaces describes interactive systems that seek to leverage natural human capabilities to communicate via speech, gesture, touch, facial expression, and other modalities, bringing more sophisticated pattern recognition and classification methods to human-computer interaction. Multimodal interfaces are growing in importance due to advances in hardware and software, the benefits that they can provide to users, and the natural fit with the increasingly ubiquitous mobile computing environment [20]

Speechweaver application combines spoken language understanding capability with direct physical manipulation in the surrounding environment supplemented by a graphical user interface (GUI). This combination gives a rich set of modalities for the users to interact in their smart environment.

3.3.2 Mobile vs Desktop vs non GUI Applications

As mentioned in [3], interaction in end user programming of smart environments would be classified into physical environment interaction and desktop/tablet based interaction. A combination of these two interaction types is needed to give better user experience when building the application. However, since computers and other devices exist in different forms and are physically distributed in a ubiquitous smart environment, applications running on a traditional desktop computer are not suitable for interaction. As mentioned in [2] ubiquitous computing environments inspire application development that is "off the desktop".

These considerations were motivation behind building a mobile device based application which enables users to move around to find various devices and their effects while defining behaviours for their smart environment.

3.3.3 Web application vs Native application

A native mobile application is developed essentially for one particular device and is installed directly onto the device itself. Users of native apps usually download them via app stores online or the app marketplace, such as the Apple App Store, the Google Play store and so on. A web application, on the other hand is basically Internet-enabled applications that are accessible via the mobile device's Web browser. They do not need to be downloaded onto the user's mobile device in order to be accessed and can run on all the devices which support web browsers.

Multimodal interfaces developed using web-standards have a number of key advantages beyond their easy accessibility to a large number of users. First, interfaces provided via the network can run computationally demanding processes such as speech recognition on fast servers, which is especially important for mobile devices. Second, web-based applications can make use of a growing array of powerful services available via the web. Further, web applications can be easily turned into hybrid applications with various toolkits (PhoneGap, Titanium Appcelerator) which package an HTML5 and JavaScript-based applications using WebViews feature in mobile devices.

Other advantages of web application compared to native application are :

- Native application require development for each mobile platform whereas web applications are developed once and run on all the platforms
- A web application updates itself where as a native application needs to constantly download updates

- Native applications are more expensive to develop and the resources are less compared to web applications.

Taking into account the above advantages, Speechweaver was chosen to be a web application instead of a native application to run on a mobile device.

3.4 End-user Application Development Analogies

Analogies are powerful cognitive mechanisms that people use to construct new knowledge from knowledge already acquired and understood [18]. When an end-user is introduced to application development in a particular domain using a well understood analogy it would be easier to comprehend the approach.

End-user development systems used analogies and metaphors to introduce people to programming and application development. Tern [9], a tangible programming language designed to introduce computer programming for children consists of a collection of wooden blocks shaped like jigsaw puzzle pieces. ‘jigsaw pieces’ metaphor is based on the familiarity evoked by the notion and the intuitive suggestion of assembly by connecting pieces together. CAMP [19] uses “magnetic poetry” as visible representations of the applications.

Other analogy which can be used to introduce general programming environment is "factory" which is described in [13]. In this analogy, A program is a factory and the learner is a factory’s creator. Factories are made of machines (programming interfaces), which are coordinated to systematically receive, manipulate, and produce products (program output and behavior). In general, learners have many tools to help create, run, and inspect their factory (the programming environment).

For Speechweaver, "cooking" analogy was used as the creation of a new smart environment application is like cooking a recipe. Cooking involves following steps :

- Deciding which recipe to cook
- Collect all the ingredients required for a recipe
- Finally, Food is cooked by following well defined step-by-step procedure by combining ingredients in varying quantity according to your taste and requirement.
- **Recipe** : A recipe is a list of behaviours that are brought together because they have something in common (Ingredients).
- **Behaviours** : A behaviour is a combination of ingredients and methods (speech , gesture, observing etc), created by end-user that produces a meaningful relation among them. In Speechweaver, behaviour takes the form of event-condition-action template.
- **Ingredients** : An Ingredient is a combination of a service and its source.

3.5 Choosing Abstractions

The author wanted Speechweaver application to be human-centric and only moderately abstract. If the implementation is too abstract it will negate users ability to reason about the program. "only moderately abstract" mean that user should be able to understand the "raw" capabilities provided by the devices and use them directly to describe environment behaviour. However, given the highly heterogeneous and sometimes unfamiliar nature of the smart environments and the devices available around them, the application should make some sense out of the components available and present to the end user. This approach removes some additional burden on the end users.

Speechweaver should provide as well as allow users to create abstractions called "templates". Templates contain slots to be filled by components. Various types of templates are possible which are discussed below.

- **Templates for activity recognition** : Various sensors values can be used to recognise an activity in an environment. This method of recognising activity is bundled and presented to the user at a higher abstraction level than the sensor data that is easy for the users to understand.
- **Templates for Task Execution** : In task execution templates, a sequence of actions can be bundled and presented as a task.
- **Templates for complete solutions** : Finally, both activity recognition and task execution can be grouped and presented to the user as a package.

Templates play a very important middle-ground between potentially inflexible applications written by developers and individual components [17]. But choosing right abstraction for a domain is very important when creating templates. In Speechweaver application, we support two types of templates : Ready made templates and User defined templates.

3.5.1 Ready Made Templates

Ready made templates are written by advanced programmers and are made available for the end users for use without worrying about how they are implemented. A ready made template can be an activity recognition template or a task execution template or a complete solution template. Templates are filled automatically when included in a recipe based on the existing ingredients in the recipe or the available ingredients in the surrounding environment.



Figure 3.1: A ready made template ingredient

3.5.2 User Defined Templates

Users can create their own templates for simple "true" or "false" end scenarios. For example, instead of describing behaviour "Switch on the lamp when illumination is less than 200", user can just say "Switch on the lamp when it is dark" by creating a template called "dark" as "It is dark when illumination is less than 200". Different combinations of sensors and conditions can be used while creating user defined templates. These user defined templates will be stored in a generic way so that user can rename it and reuse it with other components.

3.6 Collecting Ingredients (Device Referencing)

A ubiquitous smart space is usually populated by hundreds of devices that are embedded in their surroundings. They must blend into the background and provide services to the the inhabitants. The services provided by these devices are essential for the success of such smart spaces. Thus, finding and understanding these services will play a vital role in building smart environment applications by end users.

One of the key motivation behind developing Speechweaver application is to come up with a interaction approach that supports finding of available devices and their services by the end users to be later used in defining behaviour of the smart environment. This process is called collecting ingredients within the context of the Speechweaver application. There are various ways of referencing a device and its components within a smart environment using Speechweaver application and they are discussed in the following section.

3.6.1 Searching among all Connected Devices List

One way of referencing a particular device is to search among all the devices available in the surrounding environment. However, since smart spaces contains many such devices it gets very difficult to find the required device and further to confirm if the referenced device is the correct one.

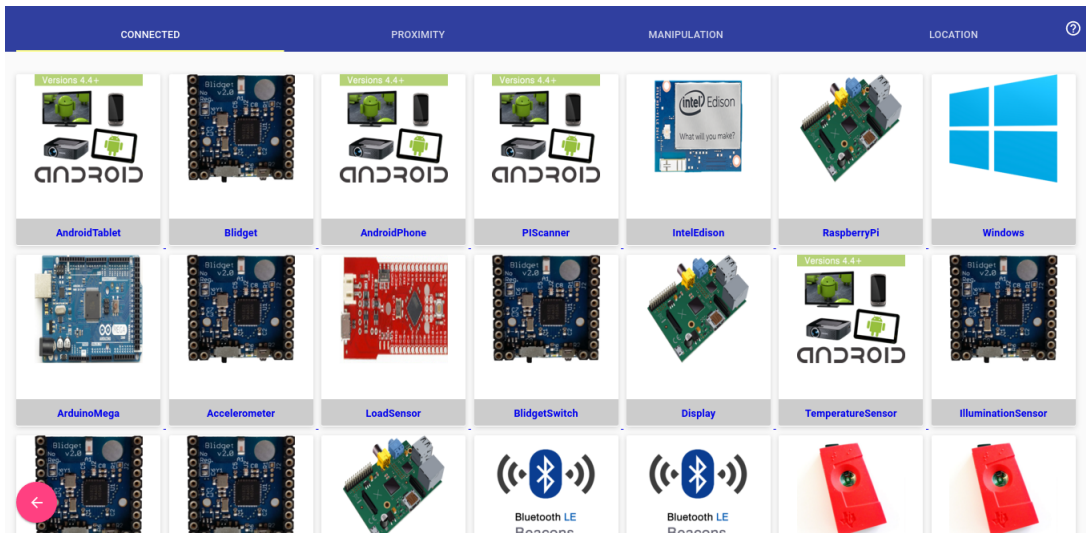


Figure 3.2: All the connected device list in a test smart environment

3.6.2 Device Referencing by Location

In this approach, devices are tagged with a location name during the initial configuration/setup of a smart environment. The location name is entered based on where the device is physically placed within the smart space. Speechweaver application lists all the locations and then when a user selects a location, all the devices whose location field is same as the selected location are listed for the user to be referenced.

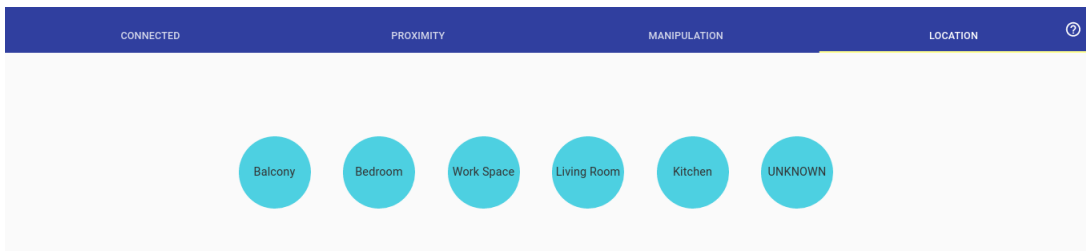


Figure 3.3: A list of all the locations to which devices are tagged

3.6.3 Device Referencing by Physical Proximity

One way to reduce the complexity in finding a device among hundreds of devices is by using the device proximity information. If only a list of devices are shown which are close to the current user then it is much easier to refer to the required device. Speechweaver provides three methods (BLE Beacons, NFC and Augmented Reality) for scanning devices which are in proximity. It takes help of other devices called "helper devices" which are other meSchup clients on which the corresponding scanning software will be running. Users can configure

helper devices in the settings options of the application and then use those devices along with the mobile device on which Speechwaever interface is running to scan the environment for referencing devices. The best metaphor to describe these interactions is that the user is “probing the world with a tool” as rightly described in [2].

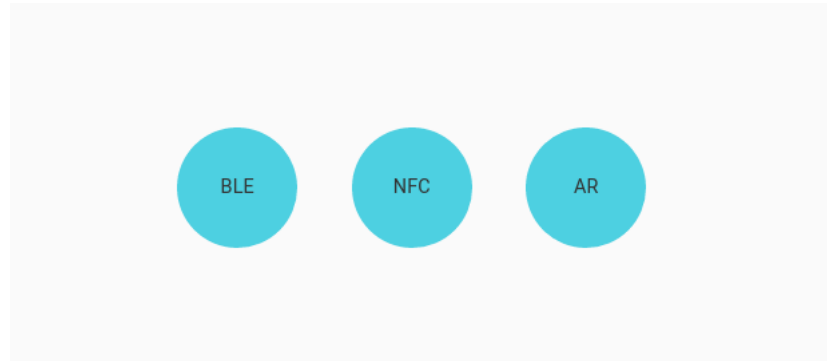


Figure 3.4: Approaches for referencing devices by proximity

BLE Beacons

In this approach, BLE Beacons are attached to the devices. Every BLE beacon has a 128-bit universally unique identifier (UUID) which is tagged to a unique device during initial configuration of the environment. By using a BLE scanner running on a smart phone, the UUID advertised by a beacon is detected and the associated device is referenced. Due to their short range, the device referencing is based on actual physical proximity.

NFC

NFC (near field communication) is a wireless technology which allows for the transfer of data such as text or numbers between two NFC enabled devices. NFC tags, for example stickers or wristbands, contain small microchips with little aerials which can store a small amount of information for transfer to another NFC device, such as a mobile phone. Each NFC tag has a 7 byte unique identifier called UID. This UID can be associated to a unique device in an environment and can be later scanned by a phone running NFC scanner to reference the device.

Augmented Reality Markers

Another way of tagging a device is by using augmented reality markers. A unique marker is associated with each device. Then by using an AR marker scanning application running on a smart phone/tablet the marker can be detected and the corresponding associated device can be referenced.

3.6.4 Device Referencing by Direct Physical Manipulation

In the direct physical manipulation approach, as the name suggests sensors are directly manipulated to reference devices to which the manipulated sensors are connected. For example, if a illumination sensor is connected to a Arduino device, then room illumination levels can be changed by turning on/off the lights or closing/opening the window shutters. In this way, the illumination sensor is manipulated and the associated device is referenced.

3.7 Content Elements

Speechweaver allows creation of different types of media content elements as shown in the figure 3.5. The content elements include images, videos, audios and web pages. They can be created by simply providing a name, URL path of the resource and selecting the type of content (image, audio, video, webpage). These content elements can then be added into the ingredients list of a recipe to be used in a behaviour. The reason for creating the media content elements is due to difficulty in referring them. For example, if the user wants to display a weather website on a screen he should specify the complete URL (www.weather.com) while creating the behaviour. But when a content element is created representing "www.weather.com", then the resource can be directly referred in the application as "weather".

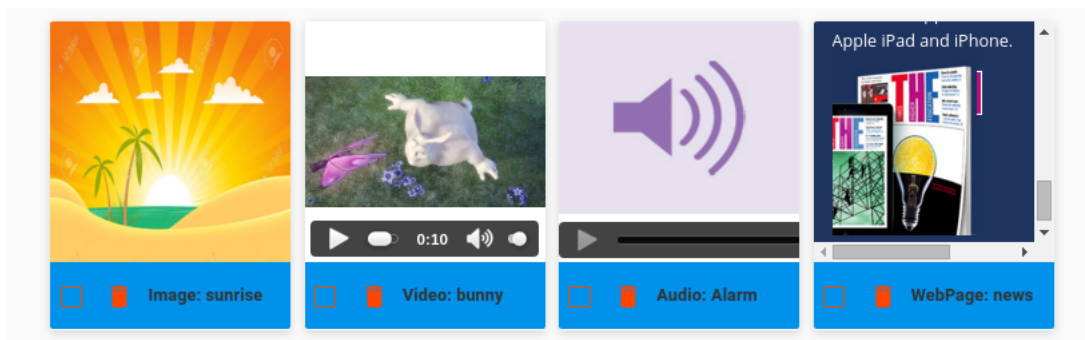
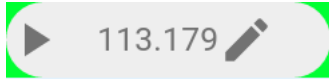


Figure 3.5: Different content elements

3.8 Live Value Element

When describing a behaviour to the Speechweaver, users can refer to the current live value of the previously referenced sensor just by saying "current value" or "present value". When a live value is referenced a special element will appear through which one can access the live value of the sensor at any point in time. Value of the live element can be frozen or set to running mode to fine tune the behaviour. There is also a manual edit mode for the live value element which enables the users to manually specify values when sensor is not currently in operation. Live element in edit mode adapts itself to the sensor type. If sensor returns a string, then in

edit mode a string input field is shown. Similarly for numeric sensor value a slider input is shown and for a boolean sensor a switch is shown.



(a) Live Value Token Running



(b) Live Value Token Freezed

Figure 3.6: Live Value Token

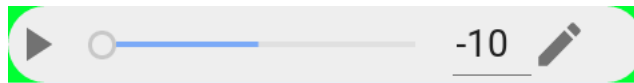


Figure 3.7: Live value token in manual edit mode

3.9 User Interface Design

User interface design of the Speechweaver should enable the following features

- Logical organisation of an IOT application using cooking analogy.
- Easily finding "things" present in their smart environment.
- Meaningful visual representation of sensor values.
- Meaningful representation of device capabilities.
- Assisting end-users by writing general purpose domain dependent templates
- Scripting cross-device behaviour by speech.
- Visualizing spoken behaviour in best way possible so as to look for errors and correcting them.

3.9.1 Behaviour editor design



Figure 3.8: Behaviour Editor Screen

Behaviour editor screen is shown in the figure 3.8. The editor screen is divided into two panels. The panel with bigger area is the ingredients panel. In this panel a list of all the added ingredients are displayed as live widgets. The ingredients representing sensors display last values sent by the associated sensor triggers. The content ingredients are represented with a bigger size widgets so that they are easily distinguished by other types of ingredients. Template ingredients can be identified by their green borders. Different kinds of ingredients can be added by clicking on the "+ Ingredients" button on the top-right corner of the main menu bar. Similarly ingredients can be removed from the ingredients panel by clicking on the recycle bin icon which turns red when in delete mode and then clicking on the ingredients to be removed. The smaller lower panel represents the cooking area which will be explained in the later sections. The interface is designed similar to traditional chat applications where upper panel displays chat messages and the lower panel displays the input area. It is designed

in this way so that the user gets the feeling that he is chatting with the application.

3.9.2 Cooking Panel (Speech Area) Design

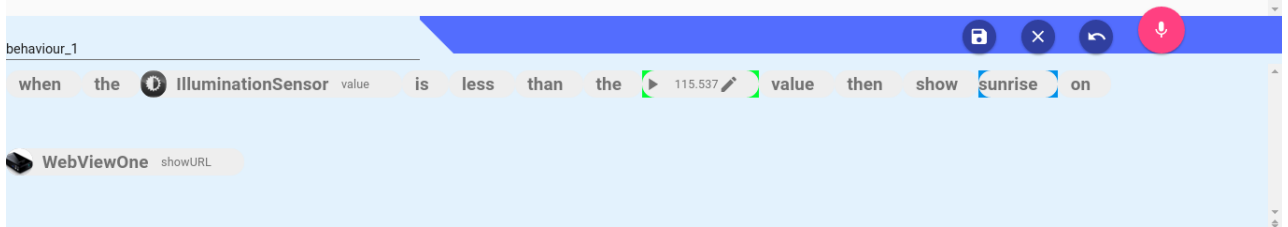


Figure 3.9: Speech area

The cooking panel is where all the speech related controls and visualization appears. On the top left corner, the input field represents the behaviour name. The relatively big button with mic icon on the top right corner is used to turn on/off the speech recognition. The second button from the right is used to remove one speech token at a time. The third button is used for clearing all the speech tokens in the speech panel. The last button is for saving the behaviour so that so that the behaviour can be deployed to the environment for testing. The final recognised speech input is displayed as tokens in the speech panel. The tokens include both plain text as well as the ingredients.

Speechweaver can understand the spoken behaviour only when certain structure is maintained in the speech. The structure is generalised to Condition-Action-Else-Action form ie. when certain conditions specified in the behaviour is met then a set of actions will be performed otherwise another set of actions will be performed.

3.9.3 Component element design

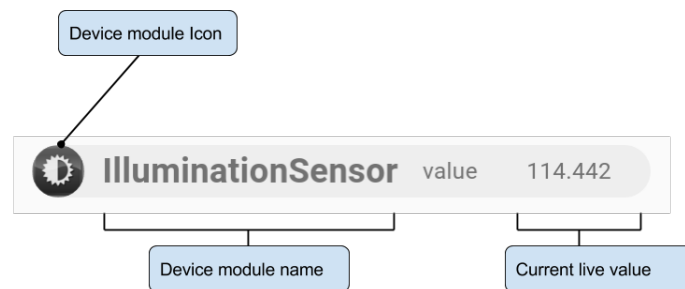


Figure 3.10: Sensor component ingredient design

Services offered by devices are visually represented in the Speechweaver as widgets as shown in the figure 3.10. The visual representation consists of an icon of the device module which

is offering the service then followed by the service name. If the service is a sensor, then the last triggered value of the sensor is also displayed as shown in the figure. Additionally, all the services provided by a module are grouped and visualized as shown in the figure 3.11

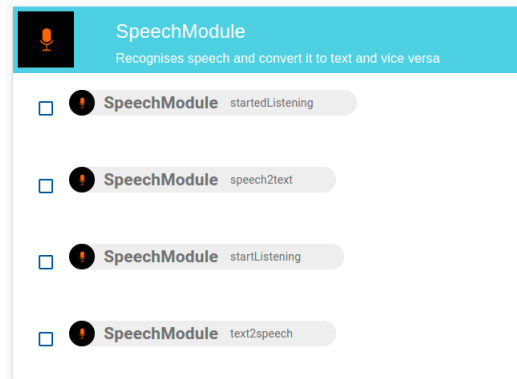


Figure 3.11: A module representation along with its capabilities

3.10 Speechweaver Spoken Sentence Structures

Even though behaviour is described by speech in natural language, Speechweaver imposes rigid structure on the spoken sentences. Basically, application can understand only one kind of behaviour description but the description can be spoken in different ways. This can be explained by an example. The basic sample behaviour description that speechweaver can understand is as below,

*"**When** the illumination is less than 200 **and** couch load is greater than 100 **than** turn on the light **otherwise** turn off the light"*

In the above behaviour description, the bold words define the structure of the spoken sentence. They are the keywords used by the application to parse the sentence correctly for code generation. Some of the keywords can be interchanged by other words supported by the application. For example, "**When**" can be interchanged with "**If**" and similarly "**than**" can be interchanged with "**do**" or "**trigger**". "**else**" can be used instead of "**otherwise**". Then the above sentence transforms to as below,

*"**if** the illumination is less than 200 **and** couch load is greater than 100 **do** turn on the light **else** turn off the light"*

In addition to using different keywords, same sentence can also have different structure but can still be understood by the system provided the system is introduced to such a structure by adding additional code generation templates which will be discussed in the next chapter. For example, the above behaviour can also be described as below,

*"Turn on the light **When** the illumination is less than 200 **and** couch load is greater than 100 **otherwise** turn off the light"*

As you can see, the sentence structure is slightly changed from the original sentence but still can be understood by the application. The behaviour description after "**otherwise**" is optional. Also, when multiple comparisons are involved in the condition part of the behaviour, "**and**" and "**or**" should be used in between comparisons with their meaning equivalent to their meaning in Boolean operations.

3.11 Application Development in Speechweaver

The entire work flow of application creation in the Speechweaver can be best explained with an example. In the following example, we will go step-by-step creating a sample application called "House Lighting" in a smart home setting. The purpose of this application is to turn on/off the lights in different rooms in the house based on the room illuminations.

Step 1 : A recipe named "House Lighting" is created.

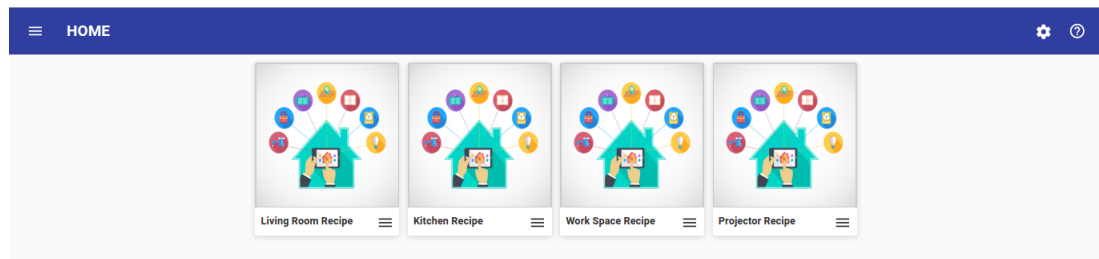


Figure 3.12: Sample Recipe list view

Step 2 : All the required ingredients for the recipe are collected. In this case, all the smart light bulbs and the illumination sensors in all the rooms should be added into the ingredients list of the recipe. The easiest way of adding all illumination sensors is by referencing by manipulation. By going to the individual rooms and turning on/off the light bulbs can trigger changes in the illumination sensor values and thus, the device to which illumination sensor is connected appears on the manipulation screen. Light bulbs can be added by manually selecting from the list of all connected devices.

Step 3 : Once all the required ingredients are added to the recipe, a behaviour for a room is created for example "living room lighting".

Step 4 : Then in the behaviour editor, a behaviour is described referring to the illumination sensor and the light bulb in the living room and saved. The sample behaviour can be as follows,

When the living room illumination is less than 200 then turn on the living room bulb otherwise turn off the living room bulb

Step 5 : Similarly, other behaviours for other rooms based on their illumination levels are created and saved.

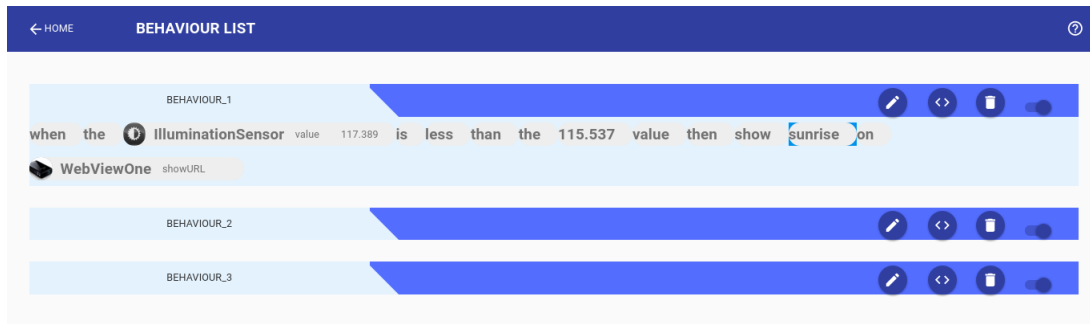


Figure 3.13: Sample Behaviour list view

All these behaviours together make a smart home application. Thus, a recipe is equivalent to an application in the smart environment.

Chapter 4

Implementation

Speechweaver is built using modern web-programming technologies, enabling the development of a browser-based application which rival the quality of traditional native interfaces, yet are available on a wide array of Internet-connected devices. It is designed to work in real-time within a smart environment. The communication between Speechweaver and the smart environment happens through meSchup middleware. Speechweaver itself is built using NodeJS on the backend and on the frontend, web components technology is used using polymer JS library. JSON files are used as database on the backend.

4.1 System Overview

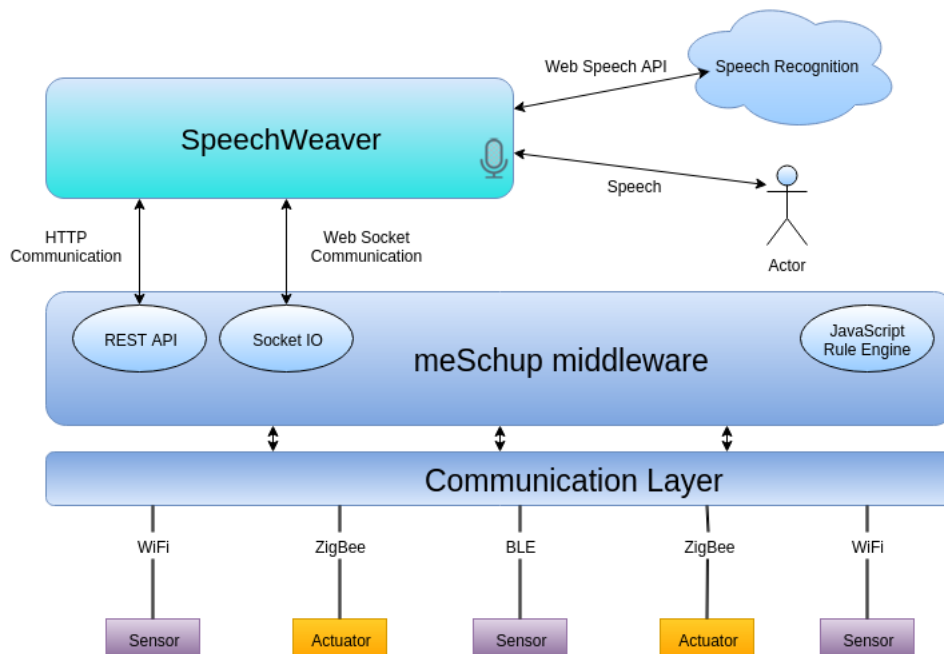


Figure 4.1: System Overview

4.2 meSchup Middleware

The meSchup middleware combines arbitrary devices, their sensors, and actuators for programming smart spaces by non experts in electronics and low-level programming. This middleware abstracts from various device communication technologies such as Wi-Fi, Bluetooth, BLE, ZigBee, Z-Wave and other incompatibilities and provides real-time access to sensors and actuators of different kind of networked device in a unified way. This reduces the time between conceptualizing a smart space interaction and realizing it to minutes instead of days or weeks, as is often the case when creating implementations from the ground up. The meSchup server runs the middleware server software. The server offers a web-based GUI for device configuration, event monitoring, and behavior scripting, and is lightweight enough to run on small, low-cost embedded devices such as the Raspberry Pi 2 or the Intel Edison. It is fully implemented in NodeJS which allows it to run on all major operating systems.

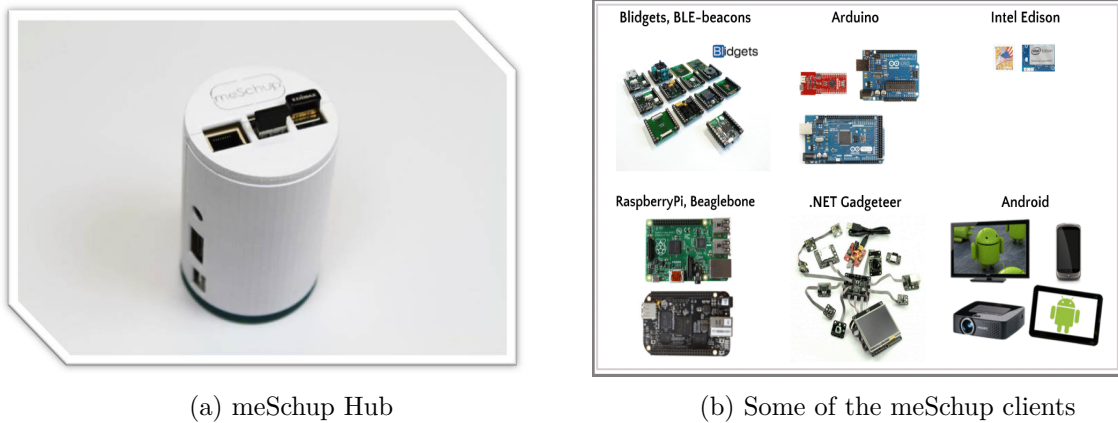


Figure 4.2: meSchup Platform

4.2.1 Rule Engine

meSchup middleware contains a scripting engine that allows users to write scripts (called "behaviours") in Javascript to weave devices, sensors, and actuators in a space into smart behavior instantly and without requiring recompilation or time-consuming redeployments. Application layer of meSchup can access any sensor in a smart environment and trigger any kind of actuator based on user-defined logic in the rules. Programming sensor-rich smart environments with Javascript opens the ubicomp domain to a huge community of web developers. Behaviour scripts are annotated with a name and a description and can be activated or deactivated anytime. Within the script editor, developer can address any device known by the middleware and any sensor or actuator in a unified way.

4.2.2 Device Ontologies

At the core of meSchup middleware, lies a knowledge base in the form of device metadata and module metadata. Device metadata provides information about device type, communication technology, device status, description and all the input and output channels available. Module metadata consists of information about associated device type, description, available settings like sampling rate and sampling interval and details about services provided by these modules like data type, data range etc. This wealth of knowledge about a device can lead to formation of device ontology. An ontology is defined as “a formal, explicit specification of a shared conceptualization” [5] and is used to represent knowledge within a domain as a set of concepts related to each other. The interrelationships of entities represented by ontologies can then be effectively used when referencing devices by speech.

4.2.3 REST API

A RESTful web API that provides the same functionality as the main GUI allows the implementation of alternative or additional graphical or multimodal user interfaces is included in the meSchup server. This API provides services like getting all the connected devices, available behaviours and modifying them.

4.3 Technologies used for Application Development

Speechweaver web application frontend is built using polymer library which is built on top of the web components standards to create custom HTML elements.

4.3.1 Web Components

Web Components [21] are a set of standards currently being produced by Google engineers as a W3C specification that allow for the creation of reusable widgets or components in web documents and web applications. The intention behind them is to bring component-based software engineering to the World Wide Web. The components model allows for encapsulation and interoperability of individual HTML elements.

Web Components consists of several separate technologies [22] :

Custom Elements : Custom elements provide a way to build own, fully-featured custom HTML tags and elements. They can have their own scripted behavior and CSS styling. This allows to give components a meaningful name so that the component can be used like any other HTML element.

HTML Templates : The HTML template element `<template>` is a mechanism for holding client-side content that is not to be rendered when a page is loaded but may subsequently be instantiated during runtime using JavaScript.

Shadow DOM : Shadow DOM provides encapsulation for the JavaScript, CSS, and templating in a Web Component. Shadow DOM makes it so these things remain separate from the DOM of the main document.

HTML Imports : HTML Imports are a way of including HTML documents in other documents. It is intended to be the packaging mechanism for Web Components. One can import an HTML file by using a <link> tag in an HTML document.

4.3.2 Polymer.js

The Polymer library [4] is a lightweight sugaring layer on top of the web components API's to help in building own web components. It adds convenient features like templating, two-way data binding and property observation to make it easy to build powerful, reusable elements with less code. The polymer library is developed by Google Inc.

4.4 Speechweaver Application Architecture

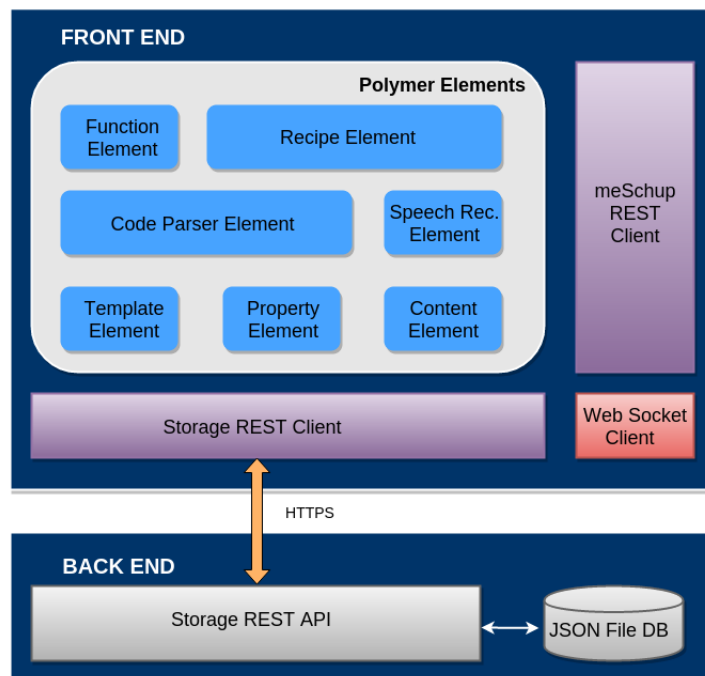


Figure 4.3: Speechweaver Application Architecture

4.4.1 Important Application Elements

Speech Element

Speech element is a wrapper for the various services provided by the Web Speech API. Web Speech API enables web applications to handle voice data easily. It has two components: Speech recognition and Speech synthesis. Speech element is a polymer element custom built for handling all the speech related activities within the application. Its API includes specifying language and dialect of the input speech and turning on/off the continuous speech input mode.

Recipe element

In speechweaver, spoken language understanding happens in two steps. The first step is taken care by the recipe element. In this step, the text returned by the speech recognition element is parsed in real time for references to the ingredients and the live value element. When a reference is found, then the plain text is replaced by the appropriate reference. The final spoken behaviour is a mix of plain strings and ingredient references. In this way, we annotate the spoken plain text with additional information which is used later for the easy of parsing code in the code-parser element.

For example, when a user describes the following behaviour : "when the illumination is less than the current value then show sunrise on the webview" the visual feedback is as shown in the figure 4.4.

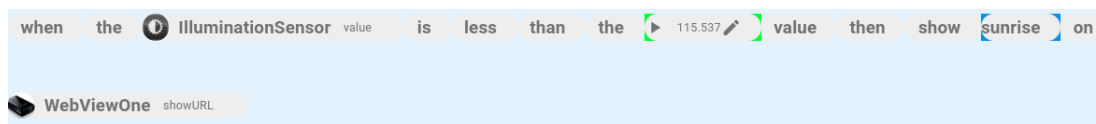


Figure 4.4: Visual feedback of annotated spoken behaviour

Code parser element

Second stage of spoken language understanding happens in the code parser element, where the annotated spoken behaviour received from the recipe element is compared against a list of code generation templates supported by the application for parsing the javascript code. A sample code generation template is shown below.

```
[
  {
    "language": "en",
    "sample": "when illumination is less than 200 turn on the light otherwise
turn off the lights",
    "vocabulary" : {
      "_if": ["if", "when"],
```

```
    "_then": ["then", "do", "trigger"],
    "_else": ["else", "otherwise"],
    "_and" : ["and"],
    "_or": ["or"],
    "_greater": ["over" , "beyond" , "large" , "above" , "outside"],
    "_smaller": ["below" , "short" , "beneath" , "within"],
    "_equalto": ["comparable" , "exact", "same"]
  },
  "condition" : {
    "_after" : "_if",
    "_before": "_then"
  },
  "action_true": {
    "_after": "_then",
    "_before": "_else"
  },
  "action_false": {
    "_after": "_else",
    "_before": null
  },
  "structure" : "_if_then_else"
}
]
```

This code generation template is used for extracting if-this-than-that-else structured behaviours. "language" key specifies ISO 639-1 Language Code of the speaker language. "Sample" field is for the reference of the structure. "vocabulary" field contains all the vocabulary used in the behaviours and their equivalent synonyms. "Condition" field specifies the code parser element where to find the "if-this" part of the if-this-than-that-else structure. "action_true" specifies the position of the actions to be taken when the condition is true and "action_false" specifies the position of actions when condition is false. "Structure" field specifies the code generation template.

Code generation element extracts behaviour structure by comparing the position of the above mentioned structural words and builds a unique string. A template is chosen for code generation whose "structure" field contains the same unique string. Code generation templates are listed in priority order so that first matched template is used for generating the Javascript code. Along with the actual code additional filters and other fine tuning code is also added in the final output. A sample code generated by the code parser is shown below.

```
// Filter 1 is for events from other sensors than ones used in the behaviour
if(! ( fromDeviceModule(api.device.Android,"IlluminationSensorOne")))
return;
// Filter 2 is for unchanged sensor values
var state_0 = load("state_0",0);
if(!( changed("state_0",api.device.Android.IlluminationSensorOne.value)))
```

```
return;
if( ( api.device.Android.IlluminationSensorOne.value < 10 )) {
  api.device.Android.photoframe.showURL = "https://www.rt.com/news/" ;
}
else {
  //else part here
  api.device.Android.photoframe.showURL = "http://worldartsme.com/images/happy-family-c
  artoon-clipart-1.jpg" ;
}
```

In meSchup rule engine, whenever there is an incoming event from a sensor, all the behaviours available in the rule engine is executed once. Due to this nature of the rule engine filters are added at the beginning of the behaviours. For the Javascript code generated by the Speechweaver application, two filters are added as shown in the above sample code. First filter allows further execution of the behaviour only if the incoming event is from one of the sensors specified in the behaviour otherwise it returns the execution control blocking further execution of the behaviour. The second filter is used for filtering out the events triggered from the sensors specified in the behaviour but when they are not unchanged from the previous cycle of execution. If the triggered sensor value is same as the previous value, then there is no state change and hence further execution of the behaviour is blocked.

Web Socket Client

Web socket client element can subscribe to a particular communication channel provided by the server. In Speechweaver, it is subscribed to channel "events". Whenever there is an incoming event from a sensor to the server, server forwards that event to all the web socket clients subscribed to "events" channel. Web socket client element then forwards these events to all the other elements within the application listening for sensor events to update their internal status.

Application Backend

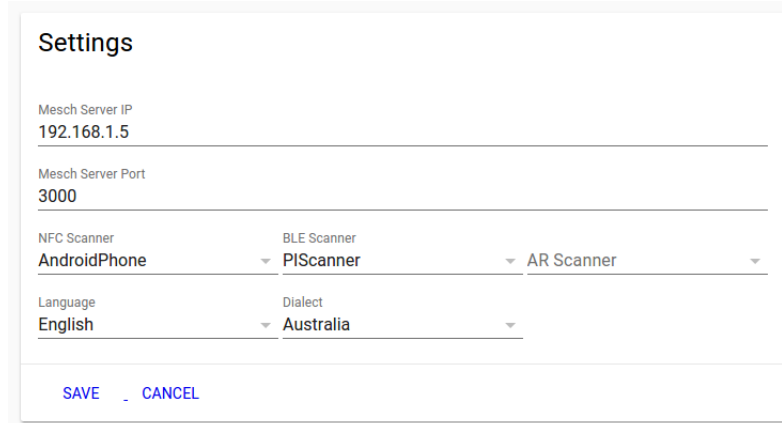
The backend of the application consists of a nodeJS server. It provides an API for managing application data stored in JSON files.

4.5 How does Speechweaver work?

4.5.1 Connecting to meSchup Server

When the Speechweaver web application is loaded in the web browser, all the web components required by the application are loaded. After all the web components are ready to be used, Speechweaver web socket client subscribes to appropriate channel ('events') on the meSchup

server socket. This enables speechweaver to receive all the sensor events from the devices connected to the meSchup hub. The IP and port address of the meSchup server required for the connection should be specified on the "settings panel" of the application as shown in the figure.



Settings		
Mesch Server IP		
192.168.1.5		
Mesch Server Port		
3000		
NFC Scanner	BLE Scanner	AR Scanner
AndroidPhone	PIScanner	
Language	Dialect	
English	Australia	
SAVE . CANCEL		

Figure 4.5: Application settings panel

After connection is established successfully between the meSchup server and the speechweaver, two rule groups ("Speechweaver" and "Speechweaver_global") are created in the meSchup server if they don't exist yet. First rule group is used for executing rules created by the application and the second rule group is used for writing ready made templates by advanced developers as described in the previous sections. Then the database of all the ready made templates and devices connected to meSchup hub is obtained in a RESTful way and parsed in the speechweaver application to represent them as templates, modules and their properties.

4.5.2 Initializing the Application

Speechweaver can be customised to the surrounding smart environment and the person using it. Back in the "settings panel", users preferred language along with the dialect (if available) is selected. Additionally, helper devices for scanning NFC tag, BLE beacons and AR marker are selected if they are available in the environment. All the available recipes and their behaviour details will be loaded from the local server database. Also, all the user generated content and user defined templates will be loaded to be used later as ingredients.

4.5.3 Recipe and Behaviour Creation

After creating a recipe and adding ingredients, a spoken behaviour described by the user for the smart environment is parsed and converted into a javascript code by the code-parser element as described in the previous section. Then, the javascript code (behaviour) is then placed in the "Speechweaver" rule group of the meSchup rule engine using meSchup REST

API. Finally, when a sensor triggers an event related to the described behaviour, it will be reflected in the smart environment.

Chapter 5

Evaluation

The aim of this evaluation is to study the user interaction experience and approaches used while programming cross-device applications and to evaluate the results. Additionally, this evaluation was intended to show that it is possible for the end-user to create new applications using speechweaver application and it's related approaches.

5.1 Evaluation Objectives

Our objectives for the evaluation were, broadly, to see how easy end users found Speechweaver for programming their environments. It does not include measuring system or the participants performance in terms of time. For the purpose of this evaluation, we focused on assessing four major questions for the interface:

- Does the Speechweaver interface allows users to describe the behaviour(program) of their connected products in a natural way ?
- Does the scripts generated by the application accurately match the user's desired behaviour?
- To what degree will the users feel the need for a visual feedback while speaking?
- Will the end users to able to understand the concept of referencing devices by various methods provided by the application ?

The focus of this evaluation was to determine the usability of the Speechweaver application and to investigate further via interviews about different approaches that participants took in order to create, change and remove behaviours from the environment.

5.2 Evaluation Methodology

We investigated the Speechweaver application in the context of smart living room with a task-based user study of 8 participants. The study was conducted during the second week of February over a period of 3 days. The average session length was 75 minutes. Data was collected by the study moderator (myself) in the form of audio and video capture, personal

interview and a set of questionnaire completed by each participant. Whole experimental set up was covered by a video camera however, option was made available to run the user study without a camera for participants who did not wish to be recorded.

Each session comprised of four phases: introduction, demo, executing tasks and an interview. The introduction phase consisted of introducing the setup and the speechweaver to the participant and were asked to fill a pre-study questionnaire. During demo phase, participants were shown a demo which involved all the approaches and functionalities that can be used for the later tasks. Then the tablet running speechweaver application was given to the participants to explore the application and get familiarised with all the options. Once participants indicated they were ready to start the tasks, the execution phase began. participants were asked to perform the three tasks by providing them with sheets containing description, sample sentences and additional keywords. The order in which the participants had to complete these tasks was the same for all (T1, T2, T3). Help was given to a participant only when absolutely necessary. Participants were allowed to complete tasks at their own pace. After completion of each task, participants were asked to give a score and comments to same set of questions related to the completed task.

After completing all the three tasks, participants were asked to fill in standard System Usability Scale (SUS) questionnaire. Additionally, another set of questionnaire related to speechweaver application were also asked to be filled.

The study session ended with a semi-structured interview. Questions in the interview were designed to elicit the participants understanding of various features, functionalities and approaches used by the speechweaver application and also to get feedback and suggestions for improving the application. Participants were also asked if they want to use such a system in their homes, work spaces and in other scenarios. Only English language was used for the user study.

Participants

Participants were recruited by posting advertisements and sending mails. Participation was voluntary. Out of 8 participants, 2 were female. Most of the participants were students at the university with one exception who was an employee also at the university. Participants age ranged from 23-41. 3 participants were very inexperienced in programming and one being highly experienced and others had a general knowledge about programming. Only one of the participant had previously used home automation products. Only 4 participants previously used speech recognition related applications before with majority specifying Apple Siri and Google Now on their smart phones.

5.3 Smart Environment Test Setup

An smart environment was setup using meSchup hub and its clients. The environment was set up at a informal place used for meetings and discussions at the HCI department in University

of Stuttgart. The smart environment consisted of a couch fitted with load sensor so that the sensor triggers events when the load on the couch seat is changed. An illumination sensor was hidden in the environment which again triggers events when the illumination level in the area is changed. Two lamps were connected to two separate edimax smart plugs which can be turned on/off by http requests from the meSchup rule engine. An accelerometer sensor was attached to the lid of a box so that when the box is open accelerometer sensor triggers events. We called this box as smart mail box. A notification device was placed on the coffee table in front of the couch which has the capabilities to vibrate, produce sound and display lights in different colors. An Android tablet was also placed on a stand on which meSchup client application with a web display was running. All the sensors are configured to appropriate levels of sensitivity. The event triggered by these sensors include the data values of the sensors.

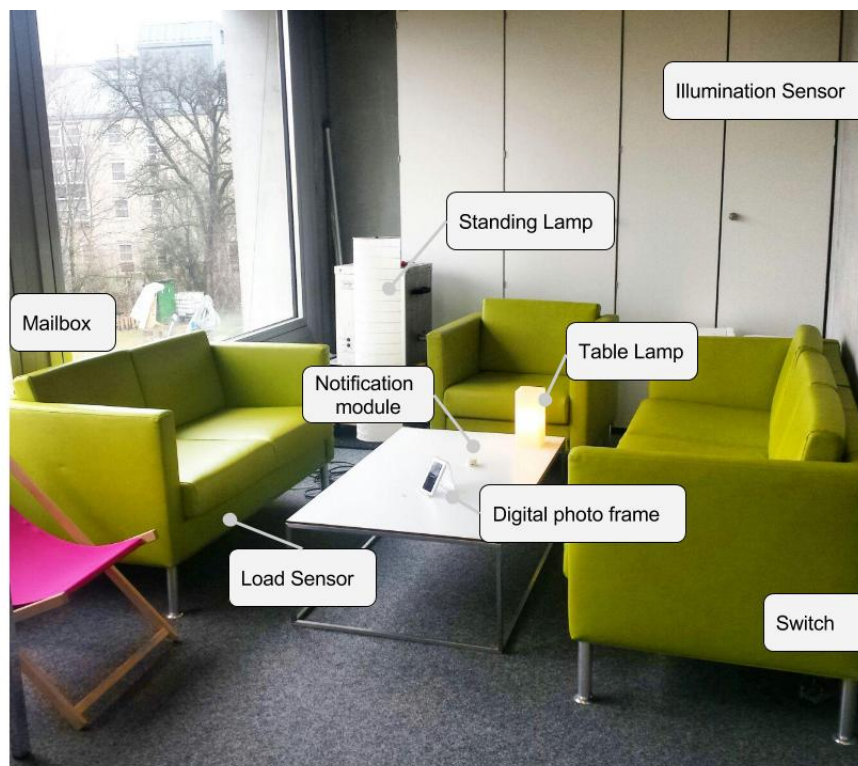


Figure 5.1: Test Smart Environment

We made ready made templates for turning on and turning off both the lamps connected to edimax smart plugs as it is not possible to do the same by speech due to involved complexity. Another ready made template called "mailbox" was created for the accelerometer attached to the lid of mail box such that the template displays "open" when the lid is opened beyond a certain point otherwise it displays "close". Finally a ready made template for notification module called "notify" was created in which it is configured to vibrate and display lights in a certain rapid frequency.

5.4 Participant Tasks

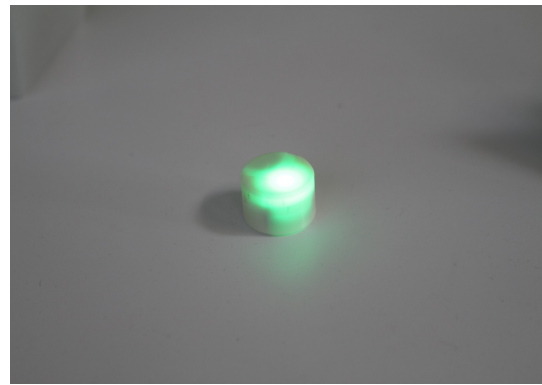
Each participant had to perform three tasks in total. First task was to introduce the concept of ready-made templates and overall approach in general. Second task included adding ingredients by various device referencing approaches and the third task combined all the previous approaches and also increased complexity which required using two conditions.

T1: mail inserted in mailbox → Notification

Usually mailbox (physical) is placed outside the house. Hence you will not know when the mailman places letters in the box or whenever someone opens the box. But you want to be notified when someone opens the mail box. Your goal is to connect the mailbox with a notification so that when someone opens the mailbox you get a notification.



(a) Mailbox with embedded accelerometer sensor



(b) Notification Module

Figure 5.2: Task 1 Smart Objects

T2: sitting on a couch → show news on the digital photo frame else family photo

You bought a new couch. It has sensors embedded in it. The sensors show different readings based on if someone is sitting on the couch or not. You also have a digital photo frame on the table which has capabilities to display photos, news, videos etc. Your goal is to relate the couch to the photo frame so that when someone sits on the couch the digital photo frame should display news. When no one is sitting on the couch it should display family photo.

T3: sitting on a couch + room lighting is low → turn on the lights

Energy consumption is a big environmental issue. You want to reduce energy consumed by your house. You see an opportunity how you can connect the smart couch to your room illumination and a lamp. You want to automate the task of turning on the lamp when room illumination levels are low. However, since you want to reduce energy consumption you want to turn on the lamp only when someone is sitting on the couch.

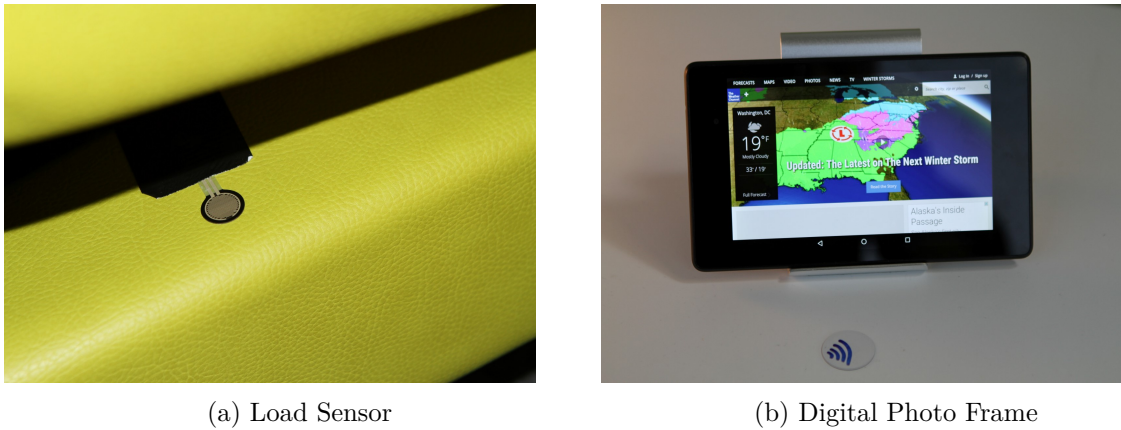


Figure 5.3: Task 2 Smart Objects

5.5 Results

All the participants were able to complete the given tasks with little or no assistance, although the time taken to accomplish these tasks varied from participant to participant.

For Task 1 (T1), participants assessed the ease of fulfilling the task on average 5.25 (SD=2.12) on a Likert scale where 1 was "Strongly disagree" and 7 was "Strongly agree". 75% of the participants were mostly satisfied with the amount of time it took for them to complete the task. 50% of the participants felt that they still need to learn a lot to be able to use the system after the task. After T1, participants like the fact that how quickly they would connect the mailbox to the notification module. They also noticed that there was no delay in the working of the system i.e there was an instant reaction from the notification module when the mailbox was opened. And they felt that the connection that they made worked well all the time without any issues. None of the participants faced significant problems with the speech recognition except that one participant complained that the system didn't understand her ascent.

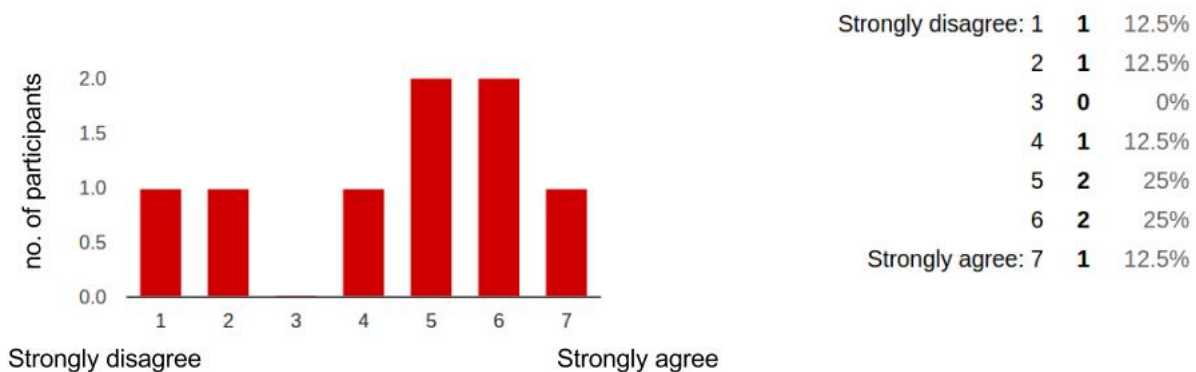


Figure 5.4: Participants rating for the statement "System had understood me well" after T1

For Task 2 (T2), participants assessed the easy of fulfilling the task on average 4.87 (SD=1.80) on the same scale as described for T1. This slight change in value indicated that T2 was slightly difficult compared to T1 for the participants. Only 50% of the participants were mostly satisfied with the amount of time it took for them to complete the task as compared to T1 which was 75%. After T2, the percentage of participants who felt the need to learn about the system increased to 75%. During T2, participants faced significant problems with the speech recognition as well as the language structure they need to use to describe the behaviour to the system for it to understand. They felt that the system imposes too many conditions on the spoken language. They also faced difficulties incorporating device names into the speech.

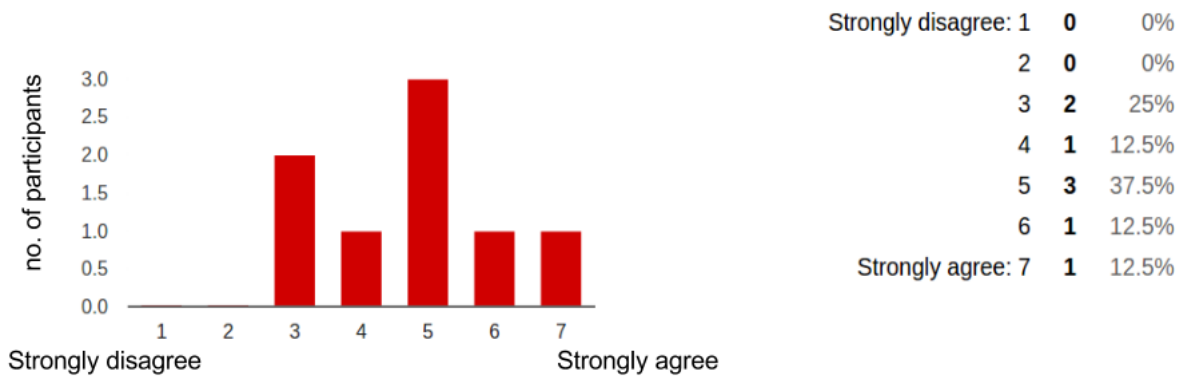


Figure 5.5: Participants rating for the statement "System had understood me well" after T2

For Task 3 (T3), participants assessed the easy of fulfilling the task on average 5.5 (SD=1.6). We interpret this increase in agreement to the fact that participants understood how the system works. 87.5% of the participants were mostly satisfied with the amount of time it took for them to complete the task. However, after T3, only 37.5% of the participants somewhat agree on the fact that they still need to learn a lot to use the system. During T3, learning effect kicked in as participants faced less issues with the speech recognition and the structure of the sentence they should use. Some appreciated that they can use multiple conditions to trigger an action. At the end of T3, some of the participants felt that it is easy to use the system and they like the approach of selecting devices.

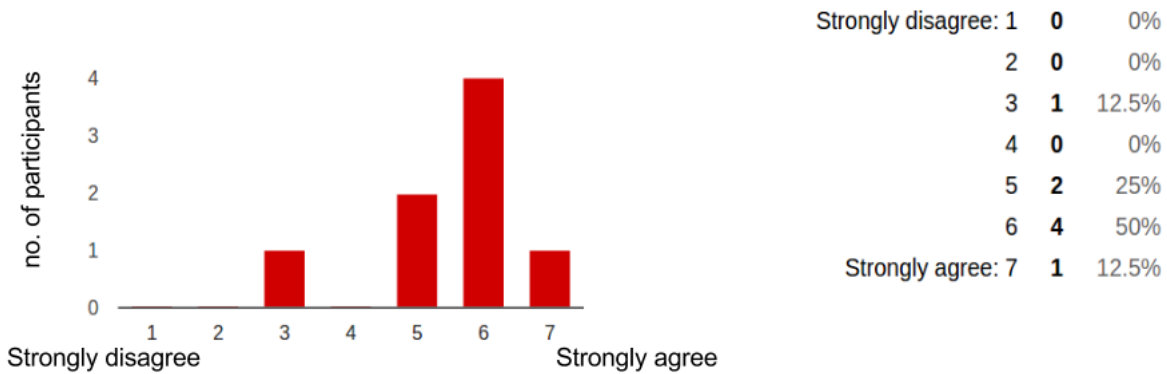


Figure 5.6: Participants rating for the statement "System had understood me well" after T3

On the System Usability Scale (SUS), Speechweaver scored 79.06(SD=14.57) which indicates that the application did not have any too serious usability deficiencies that would have effected its utility.

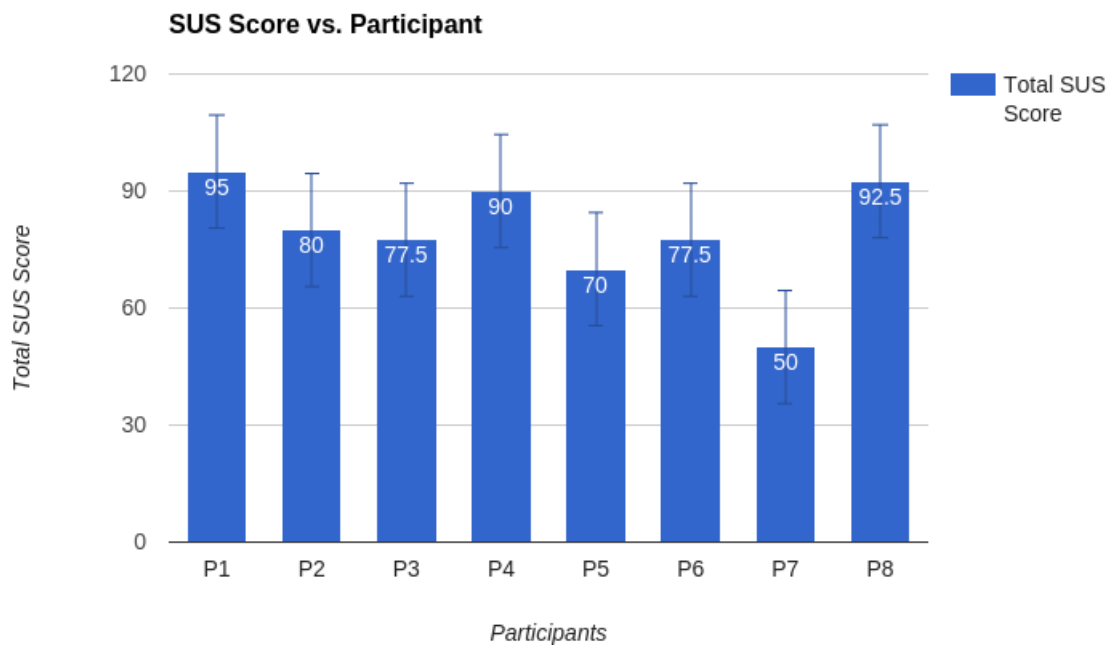


Figure 5.7: Participants SUS score

Only 5 out of total 8 participants used live value feature but 1 participant found it not so easy to use. Other choose to directly specify a value for sensor comparison. 75% of the participants strongly agree that it is important that the system works on a mobile device (tablet) but only

only 62.5% of the participants strongly agree that it is helpful that system works on a mobile device. 87.5% of the participants strongly agree that referencing devices by manipulation useful and rest were undecided. 87.5% agree using NFC tags for device referencing easy and the rest some what disagree. All the participants agree that "ingredients" concept was easy to understand and also that visual feedback while speaking is helpful. Only 62.5% of participants agree that speech recognition works well and 25% disagree about the same and rest undecided.

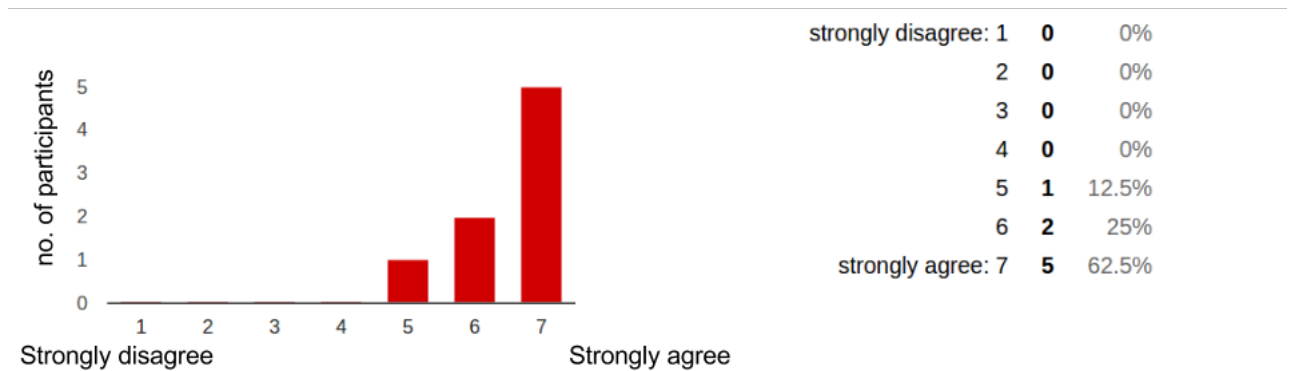


Figure 5.8: Participants rating for the statement "I found it helpful that the system works on a mobile device"

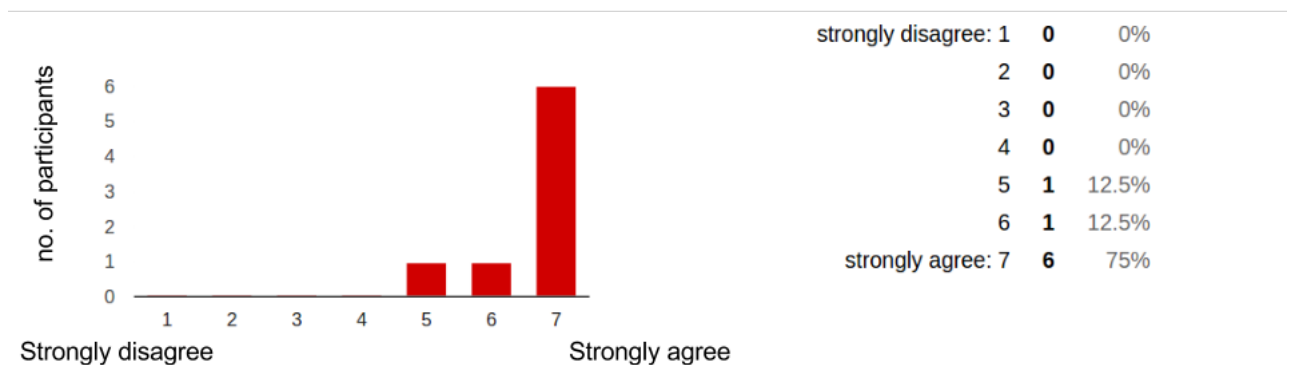


Figure 5.9: Participants rating for the statement "I find the "ingredients" concept easy to understand"

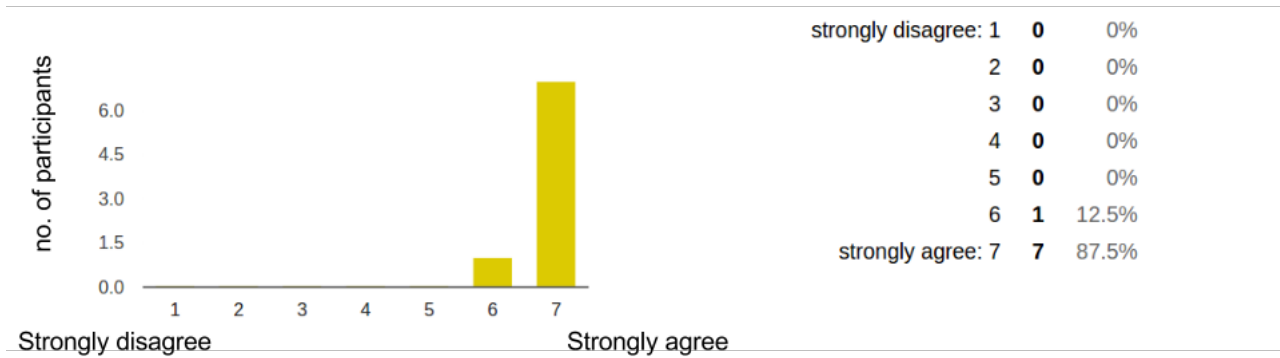


Figure 5.10: Participants rating for the statement "I found adding devices by manipulation helpful"

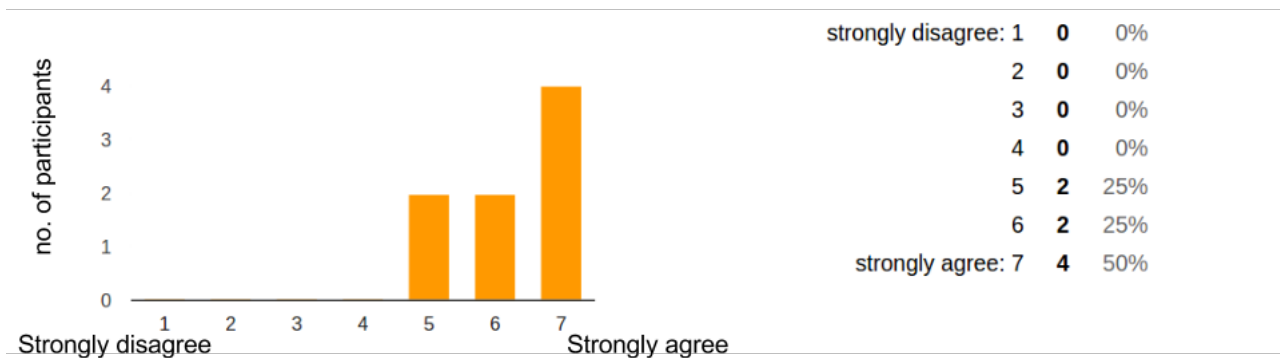


Figure 5.11: Participants rating for the statement "I found the visual feedback while speaking helpful"

The study also revealed that none of the participants found it difficult to understand the basic principles of the system. After a brief learning phase most of the participants were able to use the speechweaver to create behaviours in the smart environment.

5.6 Discussion

Even though all the participants could complete all the tasks with little or no assistance, some felt that the over all approach is very complicated. Others liked the fact that they could put many conditions into the behaviour and everyone liked the instant execution of behaviour in the environment. The other observations are discussed in the below section.

Speech Recognition

During the study, almost every participant experienced difficulty with the speech recognition results at some point or the other. But when the speech recognition results were wrong for the second straight time, the frustration levels of the participants started to increase and some participants started shouting at the tablet very loudly. One of the participant complained;

"System does not understand me"

The recognition results were unpredictable when the dialect of the English language was not chosen correctly. The recognition results was also very bad when an individual word is spoken out of context. For example, when the participant said *"when the weight is less than 20"* the recognition results were most of the time "When the rate is less than 20". But when the spoken sentence is changed to *"when my weight is less than 20"* then the recognition accuracy improved very drastically. This observation shows that we need to include complete context into the spoken speech for correct results.

Speech Visualization & Manipulation

Even though most of the participants agreed that visualization for the speech input is necessary, some participants failed to notice recognition errors while they are speaking and few didn't notice even after they finished speaking. They directly pressed "Save" button without looking into the visualization of the spoken text as one of the participant said;

I was really concentrating on what to say not what is being displayed on the screen

Participants most often forgot to both start and stop the speech recognition. This suggests that there is a need for more feedback regarding the current status of the recognition. Also, when there is an error in the recognition participants choose to clear all the text even though there was an option to clear one word at a time from the last word. Since the application shows the interim speech recognition results, some participants did not wait for the final recognised tokens when the interim results were wrong and cleared the whole text. Some participants also didn't save the behaviour and expected the behaviour to work in the environment mostly taking cue from the natural world where there is no need to confirm after one finish speaking. One of the participant did not find the visualization of the spoken text as tokens natural especially with the recognised ingredients.

Spoken Language Structure

Most of the participants did not get the structure recognised by the application correct at the first attempt. It was only after looking at the sample sentences and the keywords given that they would figure it out that they need to follow a structure when they are framing a sentence to speak. During the study, they complained that it is difficult to keep to the structure and wording, one of the participant complained as below;

"I am not used to this kind of wording and I am not used to the structure. It's more for the programming people. It's a programming speech and not a spoken speech"

Especially, they complained that "then" keyword used by the application for separating condition and actions does not fit in their spoken language style. However at the end of the study every participant felt that the approach was easy as one participant rightly mentioned

"After getting familiar with the kind of speech and its structure it is not that difficult"

Referencing Sensors

All the participants could easily relate the device ingredients with the objects in the real world and all the of them found referencing devices by manipulation interesting and very helpful. However one participant rightly said

"If you now how to manipulate something then it is easy to add them into ingredients"

Every participant sat on the couch to refer load sensor by manipulation and some of them could easily figure out that turning on the lamp will manipulate the illumination sensor to be added into ingredients. But when a sensor is manipulated, Speechweaver displays the device to which the sensor is connected this was not intuitive as participants had to go to the device view and select the sensor manually.

Templates & Live Value Element

The mailbox ready made template was understood by the participants readily when they opened and closed the lid of the box to see the template change its status from "open" to "close". But they could not understand the templates associated with actuators such as "notify" template as there is no visual or written description about its association and its capabilities. There was some confusion with the "notify" template which you can see from the participants complains below. From this observation we can infer that Participants preferred abstractions over the sensor values instead of pure sensor values but were confused when abstractions are used for actuators.

"What can notify do? I don't get that" "I didn't have the verb for what this notifier should do"

Not every participant used the live value element but the ones who used it found that it is easy to change the sensor comparison values in the behaviour later when their initial sensor observations did not produce desired behaviours in the environment. The participants who did not use the live value element where frustrated when they had to remove spoken tokens to use different sensor reading somewhere in the middle of the behaviour. One issue with the live value element is that it should be referred only after initially referring to a sensor. One of the participant did it the other way and did not get the intended results.

Chapter 6

Conclusion and Future Work

This chapter concludes this thesis report by reflecting upon the original goals and objectives of the project and presenting a detailed discussion of the overall significance of the results. This chapter will also discuss the status of the prototype and proposed future work.

6.1 Summary

The end goal of this thesis has been to come up with a best possible approach to end user programming of smart environments. We choose programming by speech as our primary approach for creating cross-device behaviours combined with demonstration of user intent during device referencing as well as during describing behaviour to the system. The application prototype was then built on top of an existing infrastructure "meSchup" which had the capability of scripting behaviour using Javascript language. In addition to programming the cross-device behaviour by speech, application also introduced various approaches that helped end users in referencing devices of interest out of multitude of available devices in a ubiquitous smart environment. Application also provided various types of abstractions over pure sensors and actuators to simplify the process for the end user. By developing such a system, we believe that we have achieved our original goal of enabling end-users create own applications for their smart environments according to their needs without relying on technicians or other programmers.

6.2 Discussion

Even though the chosen interaction approach for Speechweaver is feasible for end-user programming of smart environments, at its current stage, Speechweaver prototype requires many improvements to be made. It's serious drawback is the kind of spoken sentences required for the application to understand. The sentence structure is rigid requiring certain keywords to be included even though it is not natural in a spoken language. Other serious drawback is the lack of interactive feedback to resolve misunderstanding and seek clarification. It has lot of other drawbacks and limitations some of which are discussed below and the future work needed to improve the application is discussed in the next section.

Memorization efforts

Sometimes application forces the users to use certain keywords (ex. THEN) which doesn't feel natural for the user in his/her spoken language. This requires for the user to remember to use certain words during creating behaviour which user feel an additional burden.

Error Prevention

Application does not take any steps in preventing user from speaking language structure which is not supported. Instead, it fails silently till the user figures out the valid spoken sentence structure for creating a behaviour. there is a lack of information that should be provided to the user about the sentence structure approval or disapproval.

Suitable Feedback

The application does not point out if it understand the spoken structure or not. Further, there is no way for the user to know if the created behaviour is successfully deployed to the smart environment. The only way to confirm is, to manipulate the environment according to the created behaviour and check for the correctness of the behaviour.

If the application takes long time processing the spoken behaviour to convert it to code, the system doesn't inform user in any way about the current status and how long the user should wait.

Handling Errors

Application does not provide any error messages to solve the problem. There is no way for the user to know the right location of the error and the reason for the error and there is no communication to the users about the right actions he much take to solve the problem.

Help Tool

Application does not provide any extensive help to aid users in creating the behaviour. It doesn't provide any instructions about what kind of spoken sentences are supported. As the application is in its prototype phase, it does not provide a complete help system. The application could present more introductory information for first time users about itself.

6.3 Future Work

This section describes a number of recommendations for the future work on this project. Many improvements can be made on the application concept itself and the implementation of the system architecture and the performance. However the most important work to be done to enhance the usability of the application is to support more natural spoken language instead of rigid structures. Some other recommendations for future work are discussed below:

Location Context Awareness

The system partially addresses the location context of various devices within the environment in a very high level. But this information is not used when creating behaviours. For example, if there are multiple devices with display capability, and if the use wants to create a new behaviour for displaying news on a device located in the living room, he should just say "display news in the living room" instead of searching for display device name in the ingredients.

End-User Debugging

When the complexity of a behaviour increases it becomes increasingly difficult for the end-users to understand and identify the error source. More research on the methods and techniques to understand the behaviour and trace back the error in the design is needed to deal with the users frustration of not knowing the root cause of the issues they face.

Dialog based Error Resolution

When the application did not understand something said by the user, then it should enter into a dialogue with the user to resolve error in a step by step approach. The error resolution approach can be a combination of visual feedback, speech synthesis and device vibration.

Usable Help Guide

To enhance the usability of the application, guidance could be provided in steps to creating behaviours. This could be in the form of a wizard or simple on screen instructions advancing user from one stage to another. Another suggestion is speech synthesis of the supported sentence structures or dynamically generating meaningful behaviours from the existing ingredients and providing to the user as hints.

Avoiding Helper devices

Speechweaver uses additional helper devices (NFC, BLE and AR Scanners) for referencing other devices in the smart environment. This could be avoided by directly implementing the scanner features in the Speechweaver application itself. Implementing complete augmented reality marker scanner in a browser is highly challenging however other scanners like BLE and NFC can be easily implemented.

Behaviour Execution Visualization

Keeping track of cross-device behaviours in smart environments is a challenging task for everyday users. As the number of devices in a smart environments is expected to rise many folds, it is important to provide tools for the users to monitor, control and visualize the interactions between the devices. Even though Speechweaver application allows users to read behaviours as a plain text and also allows users to enable/disable and remove behaviours, given the number of behaviours there should be a alternate way to easily visualize the interconnections in a graphical way.

Additional programming Constructs

Currently the Javascript code generated by speech weaver contains only IF-ELSE control statements. This limitation also restricts the kind of behaviours that can created by the application. Providing support for other control structures like FOR loop will enable users to create different types of behaviour.

Appendix A

User Study

A.1 Pre-Study Questionnaire

2/28/2016

Pre-Study Questionnaire

[Edit this form](#)

Pre-Study Questionnaire

* Required

Please enter your gender *

- Female
- Male

What is your age? *

What is your occupation? *

How would you assess your general programming experience? *

1 2 3 4 5

Inexperienced very experienced

Do you have any experience with the home automation products? *

- Yes
- No

If "yes" can you mention them

How do you access you spoken English language? *

1 2 3 4 5

2/28/2016

Pre-Study Questionnaire

If "yes", how often do you use them

1 2 3 4 5

Very rarely Very often

Can you mention them

Submit

Never submit passwords through Google Forms.

Powered by

This content is neither created nor endorsed by Google.

[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)

A.2 Questionnaire for Tasks

2/28/2016

Questionnaire for Tasks

[Edit this form](#)

Questionnaire for Tasks

* Required

overall, this task was : *

1 2 3 4 5 6 7

Very easy Very difficult

"It was easy to fulfill the task" *

1 2 3 4 5 6 7

Strongly disagree Strongly agree

"The system has understood me well" *

1 2 3 4 5 6 7

Strongly disagree Strongly agree

"I was satisfied with the amount of time it took to completing this task" *

1 2 3 4 5 6 7

Strongly disagree Strongly agree

"I still need to learn a lot to be able to use the system" *

1 2 3 4 5 6 7

Strongly disagree Strongly agree

What worked well?

What did not work well?

2/28/2016

Questionnaire for Tasks



Submit

Never submit passwords through Google Forms.

Powered by

This content is neither created nor endorsed by Google.

[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)

A.3 Post Study Questionnaire

2/28/2016

Post-study Questionnaire

[Edit this form](#)

Post-study Questionnaire

* Required

"I found the speech recognition to work well" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I found the visual feedback while speaking helpful" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I find the "ingredients" concept easy to understand" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I found adding devices by manipulation helpful" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I found adding devices by manipulation easy to use" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I found adding devices via NFC tags easy to use" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I found working with the "current value" helpful"

1 2 3 4 5 6 7

strongly disagree strongly agree

2/28/2016

Post-study Questionnaire

"I found it easy to work with the "current value" feature"

1 2 3 4 5 6 7

strongly disagree strongly agree

"I found it helpful that the system works on a mobile device" *

1 2 3 4 5 6 7

strongly disagree strongly agree

"I find it important that the system works on a mobile device" *

1 2 3 4 5 6 7

strongly disagree strongly agree

Submit

Powered by

This content is neither created nor endorsed by Google.

[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)

A.4 System Usability Study Questionnaire

2/28/2016

System Usability Study

[Edit this form](#)

System Usability Study

* Required

I think that I would like to use this system frequently *

1 2 3 4 5

Strongly disagree Strongly agree

I found the system unnecessarily complex *

1 2 3 4 5

Strongly disagree Strongly agree

I thought the system was easy to use *

1 2 3 4 5

Strongly disagree Strongly agree

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

1 2 3 4 5

Strongly disagree Strongly agree

I found the various functions in this system were well integrated *

1 2 3 4 5

Strongly disagree Strongly agree

I thought there was too much inconsistency in this system *

1 2 3 4 5

Strongly disagree Strongly agree

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

1 2 3 4 5

Strongly disagree Strongly agree

2/28/2016

System Usability Study

I found the system very cumbersome to use *

1 2 3 4 5

Strongly disagree Strongly agree

I felt very confident using the system *

1 2 3 4 5

Strongly disagree Strongly agree

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

1 2 3 4 5

Strongly disagree Strongly agree

Submit

100%: You made it.

Powered by

This content is neither created nor endorsed by Google.

[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)

Bibliography

- [1] Gregory D. Abowd. “What Next, UbiComp?: Celebrating an Intellectual Disappearing Act”. In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp '12. Pittsburgh, Pennsylvania: ACM, 2012, pp. 31–40. ISBN: 978-1-4503-1224-0. DOI: 10.1145/2370216.2370222. URL: <http://doi.acm.org/10.1145/2370216.2370222>.
- [2] Gregory D. Abowd and Elizabeth D. Mynatt. “Charting Past, Present, and Future Research in Ubiquitous Computing”. In: *ACM Trans. Comput.-Hum. Interact.* 7.1 (Mar. 2000), pp. 29–58. ISSN: 1073-0516. DOI: 10.1145/344949.344988. URL: <http://doi.acm.org/10.1145/344949.344988>.
- [3] Marc Godon et al. “The First Interaction Design Pattern Library for Internet of Things User Created Applications”. In: *Human-Computer Interaction. Design and Development Approaches SE - 26* 6761 (2011), pp. 229–237. ISSN: 0302-9743. DOI: 10.1007/978-3-642-21602-2{_}26. URL: http://dx.doi.org/10.1007/978-3-642-21602-2%7B%5C_%7D26.
- [4] Google. *Polymer library*. URL: <https://www.polymer-project.org/1.0/>.
- [5] Nicola Guarino et al. “An ontology of meta-level categories”. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*. Morgan Kaufmann, 1994, pp. 270–280.
- [6] Dominique Guinard. “Current Trends in Web Engineering: 10th International Conference on Web Engineering ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers”. In: ed. by Florian Daniel and Federico Michele Facca. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. Chap. Mashing Up Your Web-Enabled Home, pp. 442–446. ISBN: 978-3-642-16985-4. DOI: 10.1007/978-3-642-16985-4_42. URL: http://dx.doi.org/10.1007/978-3-642-16985-4_42.
- [7] Valentin Heun, James Hobin, and Pattie Maes. “Reality Editor: Programming Smarter Objects”. In: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp '13 Adjunct. Zurich, Switzerland: ACM, 2013, pp. 307–310. ISBN: 978-1-4503-2215-7. DOI: 10.1145/2494091.2494185. URL: <http://doi.acm.org/10.1145/2494091.2494185>.
- [8] Michael S. Horn and Robert J. K. Jacob. “Designing Tangible Programming Languages for Classroom Use”. In: *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*. TEI '07. Baton Rouge, Louisiana: ACM, 2007, pp. 159–162. ISBN: 978-1-59593-619-6. DOI: 10.1145/1226969.1227003. URL: <http://doi.acm.org/10.1145/1226969.1227003>.

- [9] Michael S. Horn and Robert J. K. Jacob. “Tangible Programming in the Classroom with Tern”. In: *CHI '07 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '07. San Jose, CA, USA: ACM, 2007, pp. 1965–1970. ISBN: 978-1-59593-642-4. DOI: 10.1145/1240866.1240933. URL: <http://doi.acm.org/10.1145/1240866.1240933>.
- [10] C. Jones. “End user programming”. In: *Computer* 28.9 (), pp. 68–70. ISSN: 0018-9162. DOI: 10.1109/2.410158. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=410158>.
- [11] Ken Kahn. “ToonTalkTM—An Animated Programming Environment for Children”. In: *Journal of Visual Languages & Computing* 7.2 (1996), pp. 197–217. ISSN: 1045-926X. DOI: <http://dx.doi.org/10.1006/jvlc.1996.0011>. URL: <http://www.sciencedirect.com/science/article/pii/S1045926X96900117>.
- [12] Caitlin Kelleher and Randy Pausch. “Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers”. In: *ACM Comput. Surv.* 37.2 (June 2005), pp. 83–137. ISSN: 0360-0300. DOI: 10.1145/1089733.1089734. URL: <http://doi.acm.org/10.1145/1089733.1089734>.
- [13] A.J. Ko, B.A. Myers, and H.H. Aung. “Six Learning Barriers in End-User Programming Systems”. In: *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. Sept. 2004, pp. 199–206. DOI: 10.1109/VLHCC.2004.47.
- [14] Timothy S. McNerney. “From turtles to Tangible Programming Bricks: explorations in physical language design”. In: *Personal and Ubiquitous Computing* 8.5 (2004), pp. 326–337. ISSN: 1617-4917. DOI: 10.1007/s00779-004-0295-6. URL: <http://dx.doi.org/10.1007/s00779-004-0295-6>.
- [15] Timothy Scott McNerney. “Tangible programming bricks: An approach to making programming accessible to everyone”. PhD thesis. Massachusetts Institute of Technology, 1999.
- [16] Brad A. Myers, John F. Pane, and Andy Ko. “Natural Programming Languages and Environments”. In: *Commun. ACM* 47.9 (Sept. 2004), pp. 47–52. ISSN: 0001-0782. DOI: 10.1145/1015864.1015888. URL: <http://doi.acm.org/10.1145/1015864.1015888>.
- [17] Mark W. Newman et al. “Designing for Serendipity: Supporting End-user Configuration of Ubiquitous Computing Environments”. In: *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*. DIS '02. London, England: ACM, 2002, pp. 147–156. ISBN: 1-58113-515-7. DOI: 10.1145/778712.778736. URL: <http://doi.acm.org/10.1145/778712.778736>.
- [18] Alexander Repenning and C Perrone. “Programming by analogous examples”. In: *Your Wish is My Command* (2001), pp. 351–370.
- [19] K.N. Truong, E.M. Huang, and G.D. Abowd. “CAMP: A magnetic poetry interface for end-user programming of capture applications for the home”. In: *Lecture notes in computer science* (2004), pp. 143–160. ISSN: 03029743. DOI: 10.1145/1378773.1378776. URL: <http://www.springerlink.com/index/4YF49UJ57QX3FT35.pdf>.

- [20] Matthew Turk. “Multimodal interaction: A review”. In: *Pattern Recognition Letters* 36 (2014), pp. 189–195. ISSN: 0167-8655. DOI: <http://dx.doi.org/10.1016/j.patrec.2013.07.003>. URL: <http://www.sciencedirect.com/science/article/pii/S0167865513002584>.
- [21] W3C. *Web Components*. URL: <http://webcomponents.org/>.
- [22] W3C. *Web Components*. URL: https://developer.mozilla.org/en-US/docs/Web/Web_Components.
- [23] Mark Weiser. “The Computer for the 21st Century”. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 3.3 (July 1999), pp. 3–11. ISSN: 1559-1662. DOI: 10.1145/329124.329126. URL: <http://doi.acm.org/10.1145/329124.329126>.

All links were last followed on February 28, 2016.

Declaration

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

place, date, signature