Prototyping tool design

Prototyping user experience in systems with multiple devices & sensors

Joakim Bonnevier & Kristoffer Leo

Certec, Division of Rehabilitation Engineering Research

Department of Design Sciences

Faculty of Engineering LTH • Lund University • 2015

Prototyping tool design

Prototyping user experience of systems with multiple devices & sensors

Joakim Bonnevier & Kristoffer Leo

Certec, Division of Rehabilitation Engineering Research • Department of Design Sciences

Faculty of Engineering LTH • Lund University • 2015

Certec, Division of Rehabilitation Engineering research

Department of Design Sciences

Faculty of Engineering LTH, Lund University

P.O. Box 118

SE-221 00 Lund

Sweden

Acknowledgements

We would like to thank our supervisors Michael Winberg and Emil Wasberger from Topp for their valuable inputs and support throughout the project. You were always there to solve all struggles we encountered during the whole process. We also want to thank Kirsten Rassmus-Gröhn for her support on not losing the academical part of the project.

Finally a big thank to Topp and everyone on the Topp team for continuously sharing their interest and insights. We had a great time experiencing the great atmosphere you all provide each day and every week!

Malmö, May 2015

Joakim Bonnevier & Kristoffer Leo

Abstract

Many recognize the phrase Smart Home, but few have actually experienced it. The world of Internet of Things (IoT) is all over the IT industry. The hardware already exists. So, why do we not see these kinds of solutions in reality?

A simple Smart Home experience could be a system that consists of a motion sensor and a coffee machine. The motion sensor is placed above your apartment door. When it senses that you come home from work, the sensor signals your coffee machine to start brewing a cup of coffee.

The above example is easy enough for most users to imagine and design, but for them to actually create and test it is almost impossible. The skill of programming sensors to communicate with objects is still highly technical. This thesis addresses this problem and describes solutions for users with no technical background to more easily create this type of experiences.

The final prototype design of this project is a software tool design that lets users with no experience in programming create simple User Experiences that includes various sensors and multiple devices. In the prototype, these experiences can also be manually simulated in the tool without using physical sensors connected to the software.

Keywords

Internet of Things, Prototyping, User Experience

Sammanfattning

Många känner till ord som smarta hem, men de flesta har ännu inte upplevt det. Vad är det egentligen som gör hemmen smarta? Oftast pratas det om smarta prylar som är uppkopplade till internet. När en pryl väl är uppkopplad till internet kan den plötsligt börja prata med andra uppkopplade prylar och andra enheter som en mobiltelefon. Användare kan därför med hjälp av sin mobil kolla om spisen är avstängd, eller kolla så att lampan i badrummet inte är tänd. Detta är något som kallas för ”Internet of Things”, eller ”IoT”.

Ett enkelt exempel på en upplevelse i ett smart hem skulle kunna vara ett system som består av en rörelsesensor och en kaffebryggare. Rörelsesensorn är placerad ovanför din lägenhetsdörr. När sensorn känner av att du kommer hem från jobbet skickar den en signal till din kaffemaskin att börja brygga en kopp kaffe.

Hårdvaran finns redan tillgänglig. Varför ser vi inte den här typen av lösningar i verkligheten?

Om man tänker bort allt tekniskt som händer i ovanstående exempel, så är användarens upplevelse som sker tillräckligt lätt för de flesta att föreställa sig och designa, men för dem att själva faktiskt skapa och testa det är nästan omöjligt. Kunskapen om hur man programmerar sensorer till att kommunicera med objekt är fortfarande mycket teknisk avancerat.

Projektet vi genomfört behandlar detta problem och beskriver lösningar för användare utan teknisk bakgrund lättare ska kunna skapa den här typen av upplevelser.

Varför är det viktigt?

Att skapa en app för att skriva anteckningar är relativt enkelt. Appen kommer omöjligt bli omtyckt och användas av alla mobilanvändare, men möjligtvis tillräckligt många för att den ska vara värd att programmeras. När man skapar en upplevelse till ett personligt hem blir problemet däremot mycket större. Varje användare har helt olika rutiner i sitt hem, vilket givetvis är svårt att förutspå.

Det blir därför allt viktigare att själva skapandet av upplevelsen läggs hos användaren. Det företaget istället måste fokusera på är att erbjuda rätta verktyg för att göra det möjligt.

Bildprogrammering

Det finns många delar i programmering som gör det komplicerat att lära sig. Ett program består av ett antal instruktioner som behövs för att utföra en specifik uppgift. Dessa instruktioner kan skrivas på olika formellt konstruerade programmeringsspråk som alla kan omfatta olika syntax (språkuppbyggnad). Med hjälp av bildprogrammering behåller vi programmeringens grundkoncept att ge instruktioner men lyfter ut problematiken att lära sig syntax, som ofta stjäl fokus från det logiska problemet i lösningen. Detta minskar inlärningskurvan för att skapa en interaktionsbar prototyp och gör skapandeprocessen för oerfarna programmerare snabbare.

Resultat

Efter forskning kring existerande verktyg samt olika metoder av bildprogrammering har vi tagit fram en ny metod som kommer underlätta för användare utan tidigare erfarenhet av programmering. Metoden ställer inga krav på att användare ska känna till tekniska termer på sensorer eller vad de gör. Den hjälper istället användarna till att skapa upplevelser på ett språk de har större chans att förstå.

Metoden bygger på bilder av händelser som läggs i en tidsföljd. Föreställ dig en upplevelse som gör att du får ett SMS när din tvättmaskin har tvättat färdigt. För att skapa upplevelsen hade du först skapat en bild för tvättmaskinen, och ställt in den på att göra något när den har tvättat klart. I följd till bilden för tvättmaskinen hade du skapat en bild för SMS, och ställt in till vilket nummer och vilket meddelande som ska skickas. När dessa bilderna lagts ihop och tvättmaskinen tvättat klart kommer man då få ett SMS med det meddelandet man valt.

Contents

1 Introduction

The Internet of Things (IoT) phenomenon continues to grow and with this, a lack of tools for creating cool experiences with hardware that actually already exists grows with it. IoT refers to an object of any sort that is interactable, controllable or observable through the Internet. These things can be done with various sensors listening to the physical environment. An example is a system that consists of a motion sensor and a coffee machine. The motion sensor is mounted above your apartment door. When it senses that you return home from work, the sensor signals your coffee machine to prepare a cup of coffee.

A major challenge for a tool which is designed for prototyping using sensors is the vast variety of ways the sensors could communicate with the tool, as well as the long list of possible hardware to prototype with. In order to make the tool easy to use, even for non-technical users, this complexity must be hidden.

Today, IoT is a very technical area, therefore most prototypers are forced to focus on the technical connectivity between sensors, and therefore forced to neglect UX? A good prototyping tool could completely change the industry.

1.1 Background

Since development budgets are always limited, companies frequently ask about the most efficient way to generate great User Experience (UX). Would it be more worthwhile to prioritize a larger display or faster processor? How do the various product capacity really affect the user experience? These are questions every new IT project has.

There already exist various prototyping tools to prototype applications consisting one screen that can easily be used by people not knowing how to code, see examples in chapter 4. But there are no real methods for prototyping more complex systems involving sensors and multiple devices. The increase in complexity comes from sensor depending on either another sensor or another device. This is common in smart home solutions, e.g. when some person leaves the home the door will be locked and the lights turned off. The way of testing this would be to build the system and try it out in a real house with everything implemented. This is very costly and if changes have to be made it will take time to change them. Thus our main problem is to solve how to design this prototyping tool and which variables has to be taken into account for the UX of prototypes created with this tool to be as good as possible.

1.2 Goals

This thesis investigates new ways of prototyping UX when interacting with systems containing various sensors and multiple devices. In this Thesis Project we will do the following:

- Explore existing methods of UX prototyping, different types of scenarios and which sensors are of most importance for prototyping IoT UX.
- Design new ways of prototyping UX when interacting with systems containing various sensors and multiple devices.

1.3 Target group

Our tool should be flexible enough for users that are experienced with programming to be able to create anything they want without feeling limited. While it also should be possible for users with no previous programming experience to build basic prototypes.

During the project process we have identified three user profiles.

With our tool design our main focus are to get a low learning curve for *Non Programmers* and *Limited Programmers* to easily get into how the tool works. Our secondary focus is to also to fulfill high flexibility for *Experienced Programmers* to not feel limited in the tool.

Non Programmer
This is a user who does not have previous experience in programming, nor want to interact with code whilst prototyping. When prototyping with sensors they commonly create smaller data flows and will not need all functions the prototyping tool will provide.

Limited Programmer
This is a user who knows the basics of programming logic. They have a hard time creating more advance data flows, but have an easy time understanding how it works when they see a complete solution. This profile wants a tool that might be advanced, but designed in a way that makes it pedagogically easy to use. There is no need for this user to know everything at an instant; one feature at a time is enough.

Experienced Programmer
This is a user who have deeper knowledge of programming. This user is well familiar with object-oriented programming and can solve a problem using technical programming terms. They will appreciate high flexibility of the prototyping tool and might be frustrated if they find themselves limited by a programming environment.

When analyzing strength and weakness in existing prototyping tools it is important to observe the tool from the perspective of all these user types. It is vital to identify which needs of the specific target groups we would like to please.

1.4 Target prototyping tool

Just as it is important to specify which target group our tool design is intended for we need to specify which types of prototypes the tool will be intended for. Tom Söderlund, co-founder of the prototyping tool Weld states that for a user to choose from the wide range of prototyping tools existing today they first have to determine the goal of the prototype they are to produce [1]. Do you want a tool optimized for understanding the flow of a complete app, or is it more about micro-level interactions (gestures or animations)? Is the prototype intended to be shown internally within your team or to external stakeholders? Are the intended platform phones, tablets or desktop?

The tool we are designing will be adapted for Hi-Fi prototypes in medium-scope systems. The focus will be on software prototyping and the hardware part of the prototype will be on a high level. This means that it might not be optimal for prototyping large scope systems, i.e. complete applications, but to specific functionality in an UX. We will also have a high focus on fast iteration.

1.5 Related work

There are some related work that have been done in the past in this area. To try to simplify the usage of Internet of Things (IoT) for consumers. This area is somewhat new and there does not exists a lot of similar work to our as of now. However we have seen during our project that there have been an increase in articles in this area.

In a research by D. Soukaras et al [2]. they developed a suite that they called IoTSuite. The goal with the IoTSuite was to integrate this framework into tools that uses IoT. To do this integration they developed four components: an editor, a compiler, a deployment module and a runtime system. This to aid different stakeholders when developing an IoT application. For example when building an IoT system there are people with different expertise like software designer, network manager or device developers. What the IoTSuite does is that it lets these people work with their own things independent of each other and then it will be linked together and the application will be created.

Another related work is the work of D. Fogli et al [3] that developed a toolkit *sense.me* which focused to integrate social network with sensors. Their idea was that the user should create events that happens when something is triggered. Then in an implementation phase the users will connect the event to a sensor and lastly the users can test if their product to see if it works as intended.

1.6 Chapter summary

Chapter 2: Methodology
This chapter will address and explain the various methods used during this

project. There will also be a project process overview to show when these methods are applied.

Chapter 3: Exploration - Internet of Things sensors & scenarios
In this chapter we explore IoT scenarios that sensors could solve. We present ways of breaking down the possible solutions and list and evaluate which types of sensors are interesting for different kinds of scenarios.

Chapter 4: Exploration - Prototyping methods & tools
This chapter introduces methods of how to program using visual programming environments. There will also be a review of existing prototyping tools followed by a tool comparison to gain an understanding of which audience the tools are adapted for.

Chapter 5: Exploration - Sensor-to-tool integration
This chapter will give a brief explanation of how different wireless sensor communication technologies work. It will also include various sensor products that simplify the usage of sensors for prototyping. This will be culminated to an example where we try to integrate sensors into one of the explored tools in chapter 4. The conclusions of this chapter consist a list of valuable tool features used in our tool design.

Chapter 6: Valuable tool features
This chapter list some valuable features for a sensor based prototyping tool. The features are results from the exploration in Chapter 4 and 5.

Chapter 7: First design iteration - Tool paper prototype & usability test
This chapter present a paper prototype that has been designed based of the knowledge gathered from explorations in the previous chapters, especially the conclusions of Chapter 5. The paper prototype is tested on users from the three target groups.

Chapter 8: Second design iteration - Hi-Fi prototype & concept evaluation
In this chapter we design a new concept of building data flows based on the results and conclusions from chapter 7. We also create a Hi-Fi prototype to test and evaluate whether the new concept is easier to understand.

Chapter 9: Third design iteration - Simplify sensor names & refining concept design
In this chapter we discuss how to name and categorize sensors in a more intuitive way. There will also be some visual and functional changes to the tool concept that will be evaluated in a heuristic evaluation.

Chapter 10: Fourth design iteration - Sensor visualization & complexity levels
This chapter expands our concept to higher complexity prototyping. We will

also present a concept for better visual representation of how the sensors used in the data flow works.

Chapter 11: Discussion
This chapter is a summary of discussions and conclusions made in Chapter 7 to 10. It also discusses why our solution is relevant for the industry.

2 Methodology

This chapter will address and explain the various methods used during this project. There will also be a project process overview to show when these methods are applied.

2.1 Project process methodology

Figure 2.1 gives an overview of the work process used in this project. The four design iterations have different approaches in the testing. How the design is evaluated will be explained in the remaining of this chapter.

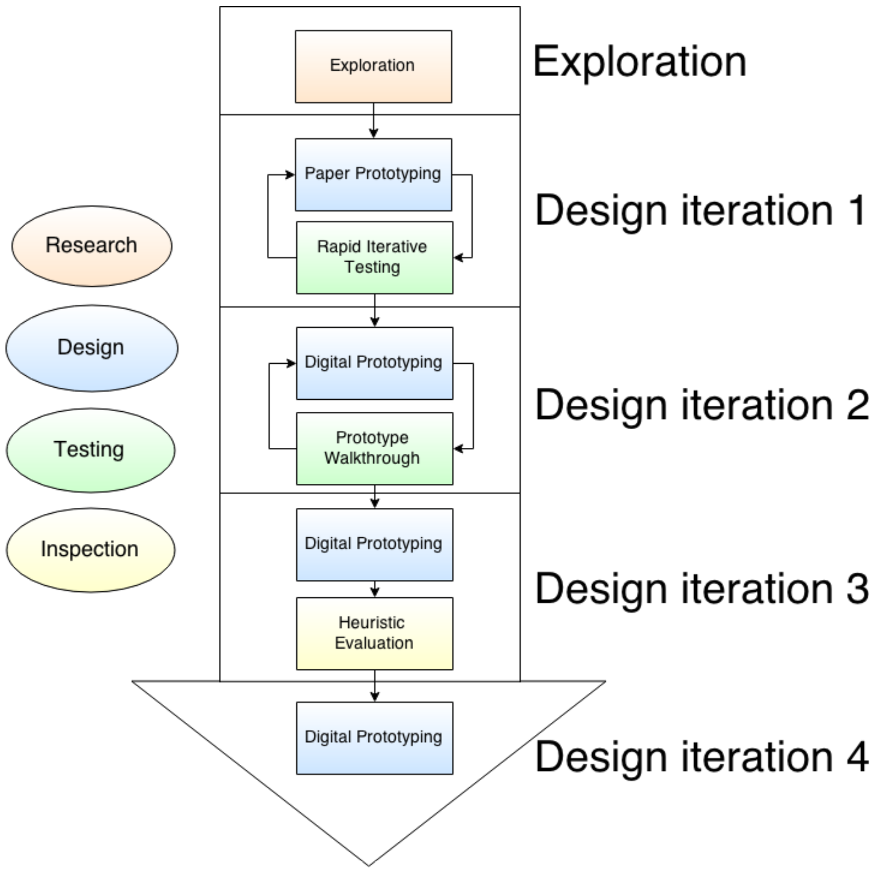


Figure 2.1: A representation of the work process

2.2 Plan, do, check, act

The problem of this thesis was how to design a prototyping tool with various sensors implemented without aggravating the UX. As a start there was a research phase where different prototyping tools were tested to get a good basis on which solutions are good and which are bad and also why they are that. At this phase a literature study was conducted, in order to create a solid foundation around the subject.

During the project, an agile method was used to develop a good tool. A plan, do, check, act (PDCA) cycle was used to get better result in the end. The PDCA [4] cycle is an agile method that can be integrated in almost all types of projects. It consists of four different "phases":

Plan: In this phase the objective of the project are made clear and how to implement them.
Do: This phase is the developing phase, here you fulfill the goals and objectives that has been set in the planning stage.
Check: In the check stage you evaluate the results you get from the do stage. And since this cycle is an iterative process you compare the results now to the earlier results to see how to add the new results into the information gathered.
Act: This is the last phase in this cycle. This phase acts as a summarization of the other phases, what to think about, what has been learned and how to optimize the next iteration of the cycle.

2.3 Rapid iterative testing

Rapid iterative testing and evaluation (RITE) is a testing method that originates from a report where a developer team for the tutorial of Age of Empire II tried to make changes quick to make sure everyone could go through the tutorial without any trouble [5]. In RITE you test your program with users, and as soon as the user run into some trouble that might be a big fundamental flaw you may change it. Then you test your new version and see if your solution solved the problem or if it added more confusion or if a new problem appeared. This puts some pressure on the test leader that must grade the problem and errors the user finds and how urgent the fix is (if only one person has problem with a small thing, then it might not be worth changing). The time it takes to make changes also play a big part when using these kind of tests, they should be able to be done quickly otherwise they might not be made at all. When testing Age of Empires II [5], they listed the issues and graded them in four different categories

1. Obvious fast fixes.
2. Obvious but a bit bigger fixes that are sent to be implemented.
3. No real obvious cause and no instant solution.
4. Failures due to other things for example test scripts or interaction with the participant.

The first two category issues were solved as fast as possible and then put into the test to see if it fixed the problem. For 3 and 4, they tried to collect more data to see if you could somehow change them to first or second category.

Eventually the error rate will go down as you try to solve the issues and hopefully by the end of the tests the participants will not have any problem with solving the task or meet the required knowledge base. In the Age of Empire II study they reached a stage where there were no or small errors made for 4-5 users in a row. They reached this stage after only 15 users in total had made the test. After ten users had done their testing they had made six changes from the original.

RITE is a great method of testing when you want to try out the main features of a program and do quick changes to the program to get fast results. This type of testing puts some pressure on the test teams to categorize the significance of the errors made by the participants and see if this is a problem many will have trouble with. However you do not have to test a lot of participants to get a desired result.

2.4 Prototype walkthrough

The setup for a prototype walkthrough is very similar to RITE. There is a scenario to evaluate if the prototype is handled as intended. The prototype walkthrough is more about the feeling the user gets and how they use the system. When performing a prototype walkthrough, the questions asked should be more about why they do in a specific way and how they would like it to be. Especially the questions after the test should be open ended, to promote more of a discussion with the tested person and give them an opportunity to express what they think about. It is as concerned with fast changes as the RITE method but can still be used to see if the design of the prototype is good for the target group [6].

2.5 Heuristic evaluation

Heuristic evaluation is an informal evaluation method that focuses on finding errors in an interactive graphical design. To perform a heuristic evaluation you let people test a system and focus on ten key heuristics (see appendix B) to make an evaluation. These heuristics will let the evaluators find errors in the usability (if there are any), which they can point out to the designers. Everyone could do these evaluations since the heuristics are quite simplified and people with different areas of expertise might find different errors.

According to the article by Nielsen and Molich [7] they take note of that heuristic evaluation is hard. From their study the evaluators found 51-20% of the usability errors in their four tests but they also say that this is much better than not finding any errors at all. The findings are often the most obvious ones and the ones that are most crucial to fix.

Nielsen and Molich [7] conclude that heuristic evaluation is not well suited to perform when it is only one person that are evaluating and recommend that using three to five persons for best results. Using more than five persons will not increase the number of error find by that much, to increase the number of errors different evaluation methods could complement it.

Nielsen and Molich [7] lists four advantages of the heuristic evaluation as following:

1. It is cheap.
2. It is intuitive and it is easy to motivate people to do it.
3. It does not require advance planning.
4. It can be used early in the development process.

The downside of this evaluation is that it only finds errors but does not suggest any type of solutions to the errors that are found. This is mainly because the persons performing the evaluation is focused on the heuristics and finding errors according to them and not how to solve the errors [7].

3 Exploration - Internet of Things sensors & scenarios

In this chapter we explore opportunities that sensors could ease ones everyday life. To come up with practical IoT solutions can be difficult, seeing that the possibility of data collection from objects around you is for most people a completely new way of thinking. We present ways of breaking down the possible solutions and list and evaluate which types of sensors are interesting for different kinds of scenarios.

3.1 Introduction to Internet of things

Internet of Things (IoT) is a phenomenon that has become more and more widespread for "the common people". What it refers to is simply a device of some sort that interacts and can be controlled with through Internet. It can be everything from a small temperature sensor to a large complex machine.

Sensors have been around for a very long time, take for example a thermometer, it was invented in the beginning of the 17th century. But to use them through internet and make them interact with other sensors is another story. The term IoT was probably first used in a presentation by Kevin Ashton in 1999 that had the title Internet of Things [8], this presentation was about the then new phenomena RFID. But it was not until somewhere between 2008-2009 "IoT was born" according to Cisco IBSG (Internet Business Solution Group) [9]. Their definition of when IoT was born is when devices connected to Internet exceeded the population of the earth.

At the beginning of the 21st century when Arduino [10] and Raspberry Pi [11] became well known and easy to use, the gap between people without any real knowledge about hardware and professional hardware programmers shrunk. Arduino and Raspberry Pis' free easy-to-use software made it easier and a lot cheaper for hobbyist, students and teachers to learn and work with simple sensors and make them interact with each other. This was one of the seeds that lead to a very big community around IoT [12][13]. In 2012 a new concept, Twine [14], was born out of a Kickstarter project. Their ideology was that everyone should be able to use various sensors and be able to connect them easy through their hardware. This resulted in software that works more or less the same way as If This Then That (IFTTT, see Chapter 4.2.6). IFTTT was at a very early stage at this point and had only launched a few months prior to Twine and they had probably taken inspiration from each other or had the same mindset. As a result of all this in 2014 IoT was at the top of Gartner's "Hype cycle of technologies" [15].

The usage domains of IoT are wide and contains everything from personal use in the home, to machines talking to each other in industries. This makes it vulnerable for outside threats that want to sabotage or steal valuable information. To understand how the security in IoT systems should work an article about it was presented at the Norwegian information security conference (NISK) [16]. In the article they state that The Telecommunication Standardization Sector of the International Telecommunication Union (ITU-T) recommends using four layers similar to the OSI model to represent the abstraction layers of IoT systems [17]. These four layers are device layer, network layer, service support and application support layer and application layer. To get a good protection from outside threats all these layers must be protected, and in NISK report [16] they suggested to use an adaptive security approach to solve the problem. Adaptive security is a security system that adapts without human interference, to what happens to the program [18]. With an interchanging security system that are modified for each layer the possibility for malicious threats are decreased significantly. For an average person this high level of security is best used when dealing with private matters for example in healthcare and specifically eHealth were IoT systems are applied security is of most importance [19].

3.2 Smart scenario domains

To narrow down and map which types of sensors that are interesting from a prototyping perspective we came up with a few IoT scenarios that we would like to prototype. Thereafter we establish which sensors that could be used in specific for these scenarios. When listing possible sensor solutions it is important to not be restricted by solutions that will work in “real life systems”, but to also include “rough” solutions that only work approximately correct to prove a concept.

It is hard to get a grip on exactly which new possibilities sensors could give to your everyday life. To get a better overview of what you are able to do you could categorize the area into smaller application domains [20], see figure 3.1. These in turn can be narrowed down to more detailed functionality which possibly will inspire you to new areas you have not been exploring before.

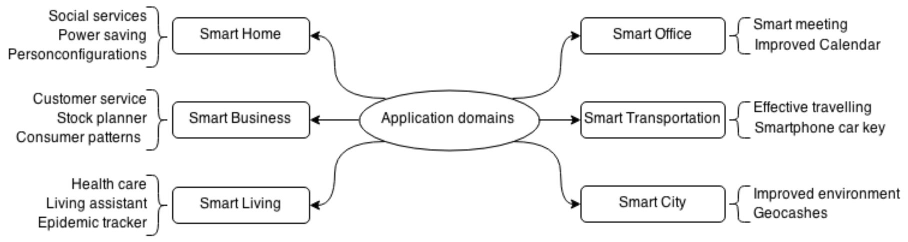


Figure 3.1: Application domains [9]

3.2.1 Scenarios

The scenarios in this section were developed by us to get a feeling for how a person can use sensors in smart home solutions. These scenarios were the base for the sensor analysis later in this chapter. The scenarios were based on our imagination and inspired by various movies and home pages such as [21].

Title: **Responsive lights**
Type: *Smart Home*
You are relaxing in front of the television in your living room. The living room lights are slightly dimmed and in the rest of your apartment they are turned off. Suddenly you get a craving for a meatball sandwich. You stand up and walk towards the kitchen. When you enter the kitchen the lights are already on and you start preparing your sandwich. In the meantime the living room knows that you left the room but also knows that you are only going to be gone for a short while. It kept the lights dimmed and paused the movie you are watching.

Title: **Forgotten card**
Type: *Smart Home*
The time is 8:02 am and you are already late to work. You finish brushing your teeth; throw your computer in the bag and half running walk to the car. When you closed the door behind you, your house sensed that you were the last person to leave and locks the door together with alarming the first floor. It also sense that you forgot to pack the company card you need to enter your office and sends a warning to your smartwatch. You notice the warning before even starting your car, sighs, and walk back to fetch the card.

Title: **Freezer door**
Type: *Smart Home*
It is evening and you are mentally tired after a long day of intense meetings. You drag yourself to the kitchen to scoop some well-deserved ice cream. When going back to the living room you did not think of closing the door of the freezer. After a short period of time you get a notification telling you to close the door.

Title: **Milk dilemma**
Type: *Smart Home*
You are in your local grocery store shopping for your weekly needs. Before you left you did a quick eye scan over what to buy and, for instance, found that you have enough milk for cooking the pancakes you planned for the evening. While you are away shopping, your roommate opens the fridge and pours herself a big glass of milk, mixing it with chocolate powder. Knowing you are away shopping, the fridge senses that the milk levels suddenly became too low for your evening pancakes and sends you a notification to buy more milk.

Title: **Customer interest**
Type: *Smart Business*

You are the owner of a bookstore with a stock of 859 different book titles. In your store you have in particular one bookshelf where you put books you want to highlight for the customers to instantly find what they might be looking for. When a customer browse and pick up a non-highlighted book your smart store will sense and register which books are getting interest by being picked up and down. At the start of each day you are checking the data from the day before to find out if you should arrange your highlighted bookshelf to help the customers with what they are looking for and a possibility to be increasing your sale rates.

Title: **Custom experience**
Type: *Smart Business*
You and your friend decide on taking a trip to the museum located in an old castle in your city. Neither of you have been there before. You do not really care for history and information of the artifacts, but enjoy the sentimental atmosphere this old castle has. Your friend on the contrary is a history junkie and reads every word of information that could be found. When you arrive to pay for the entrance you are both given an application in which you can state which type of experience you want to experience. You enjoy sound effects vivid stories so you chose to enter in horror mode, and your friend that doesn't care for that as much and want straight facts chose informative mode. When you and your friend start the route, the application senses in which room you are and subsequently presents different stories, information and various events depending on where you are and which artifacts you are currently watching.

Title: **Quick service**
Type: *Smart Business*
You're at the café, reading Business Insider and to get some of you emails out of the way. It has been a while since you took your last sip of the coffee you bought. Suddenly you receive a popup on your smartwatch telling you that your coffee has become cold and asking you if you would like a refill. You are going to stay for at least 30 more minutes and press yes, which sends an order to the barista who soon arrive to refill your mug.

Title: **Morning run**
Type: *Smart Living*
It is Sunday morning, which is the time you normally go for a 10km run. You put on your running clothes, shoes, headphones and start on one of your favorite music playlist on your media device. You start by power walking the first 800 meters to soften your muscles. During this period a slow beat track plays on your music device. As you start jogging your media device sense that your tempo is increasing and queue a faster bps song.

Title: **Optimized sleeping**
Type: *Smart Living*
You are currently in a stressful period of life and you have started to experience worse sleep patterns then you normally have. You feel like a wreck each

time you wake up and you don't really know why. Your smartbed noticed this new behavior and begin waking you up a bit earlier when you are in light sleep instead of deep sleep, which gives you the feeling of being more rested when waking up.

Title: **Canceled meeting**
Type: *Smart Office*
It is Wednesday and you have a 5am conference call scheduled with one of your colleagues from abroad. During the time you are asleep your colleague suddenly got indisposed and have to cancel the meeting. Your corporate email communicates the changes to your alarm clock, which resets itself to wake you up the time you normally wake up.

Title: **Free cab**
Type: *Smart Transportation*
It is Friday and you have an important business meeting registered at 2pm in your calendar. 15 minutes prior to the time you need to leave your calendar orders a smartcab to arrive outside your office just in time. The cab is self-driven and filled with advertising screens. On one of the screens you see an ad for the jacket you tested in a store yesterday but had to think through before buying. You decide on buying the jacket and send out an order. The screens also sense that your partners' birthday is coming up in two weeks and presents alternatives that might be interesting to give for present. The cab ride itself is free of charge.

Title: **Bus tracker**
Type: *Smart Transportation*
You are normally traveling home by bus and have the possibility to take different busses with different route, that gets you home in about the same time. In your smartphone you can receive a live update over where the interesting busses are, how late they might be and how many passengers they already have. You will almost never have to stand up in a bus again.

3.3 Sensors for prototyping

A sensor is a small device that can gather information about its surrounding depending on which type of sensor it is. It is also a device that trigger events when something interacts with it. Normally a sensor has one activity to gather information about, for example a temperature sensor will measure the temperature in its surroundings.

3.3.1 Sensor types

Here the most common and useful sensors are briefly explained.

Accelerometer
Measures the gravitation, $meter/second^2$. E.g. if the sensor are not moving

and starts to move.

Gyroscope
Measures orientation of the sensor if it is up/down or sideways. E.g. if the sensor if you use the sensor in a plane it can tell if the plane is upside down or not.

Force sensor
Measures the pressure applied to the sensor. E.g. it can take notice if something is placed on the sensor or not, as well as how much it weights.

Motion detector
Detects if something moving in its view. E.g. someone enters a room you can have the sensor placed at the doorway and when a person crosses the door it will notice it.

RFID tag
A small chip that can transfer information that are programmed on the chips to a reader. E.g. in a bus where you have an RFID-chip in the card and when you can it is programmer to reduce money from the card.

Bluetooth beacon
Devices that can work as a trigger for actions when a smart device such as a phone or a tablet gets close enough to the beacon. The most common beacon is the iBeacon, which has become a normal term for this device. **Light sensor**
A sensor that senses the intensity of the light. E.g. if it is dark or bright.

Audio in
Audio in is something that detects the audios in the vicinity.

GPS
Communicates with satellites to get the position of the sensor, can also calculate velocity due to movement derivation.

Thermometer
Measures the temperature at the sensor. E.g. how cold it is in a room.

Barometer
Measures the air pressure around the sensor. Is a normal way to determine how high you are above the sea level.

Hygrometer
Measures the humidity level around the sensor. Often used to regulate the humidity level in different places where too low/high humidity can be damaging.

Magnetometer

Normally measure the magnetic field of the earth and works as a compass but can also measure the magnetic force from a material.

Camera
Can take pictures and also record videos. Can be used to recognize faces depending on how advanced it is.

LED
Emits light when a given voltage is applied to it. E.g. normal lights can be replaced with LED, which normally have a broader color spectra than a normal light bulb.

Servo motor
A motor that can control its position, velocity and acceleration. E.g. a robot arm that can do fine motions.

3.4 Sensor evaluation based on scenarios

To evaluate the sensors in the scenarios we listed the most common sensors on post-its to get an overview over them. The sensors had to be categorized in some way to easier evaluate them. We found some ways of categorizing them, but found that categorizing them by, for example, acoustic or magnetic was not what we wanted [22]. Instead we tried to find an own way of categorizing them that were more applicable to our needs. The categorization that was used was based on input/outputs since we found it to be a good way to separate them when comparing them in the scenarios.

In the scenarios there were quite a few outputs and in general there are more input sensors than output sensors. Because of this we decided to divide the inputs into trigger listeners and sharp triggers.

The trigger listeners are sensors that send information to a device that receives the information and does something with the information. For example you have a temperature sensor that measure the temperature outside the house all the time, if the temperature drops below a certain degree you get a notification that it is cold outside you might want to put on a jacket. The sharp triggers are more sensors that do something when something happens to the sensor. These work more or less like geocachers where something happens approach the area where the geocacher is. The main thing with the sharp triggers is that they only do something during a short time and not over a long time as the trigger listeners do. Normally the sharp triggers have on/off or true/false statements whilst trigger listeners measures values.

Some of these sensors can be inserted into both the trigger listener and the sharp triggers categorizes since they have a very wide spectrum of usage.

When listing the sensors we think about them as standalone sensors and not for example the ones you have in the mobile phone. This mainly to separate them from each other. Another important aspect is that they become more versatile in their usage if they are not limited to any other device.

To be able to do some kind of comparison between the sensors, some kind of measurement of how useful they are had to be made. We found that a simple point system mapping could be useful. When studying the scenarios we thought about how the scenarios would be realized and which sensors could be used. If they were used in a good way we put one point in the column and if they can be used but it is kind of farfetched we simply added a half point. The results of this are described in the graphs in the next subchapter.

3.5 Commonly used sensors

As passive listeners for our scenarios we found that Gyroscope, Accelerometer and iBeacons are most useful. Barometer, Light sensor, Audio in and RFID are used rarely. Magnetometer is not used at all in our scenarios.

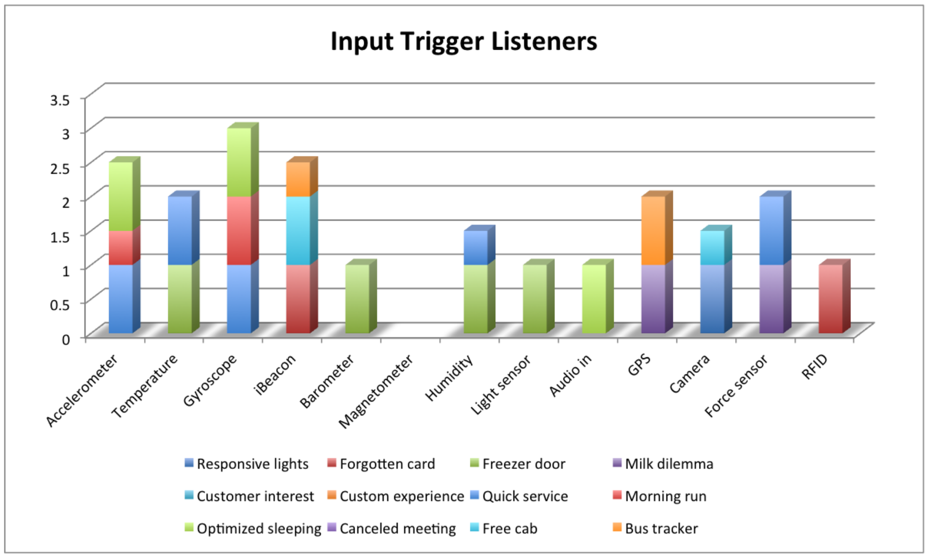


Table 3.1: Input trigger listeners.

For the scenarios we have chosen you can see in table 3.2 that iBeacons are most commonly used. Slightly after comes Motion detector. Gyroscope, Accelerometer and Button are used rarely

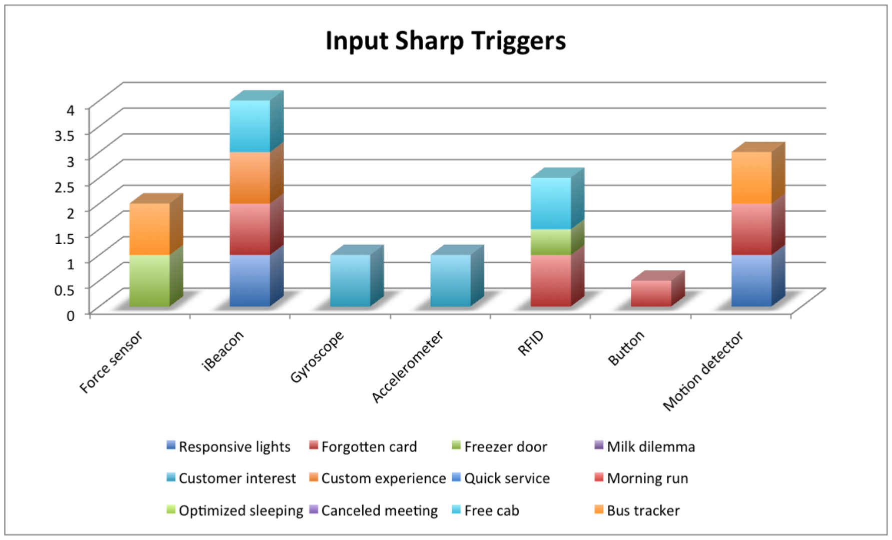


Table 3.2: Input sharp triggers.

Neither of the output sensors are used frequently in our scenarios. LED and Servo motor are slightly more usable than Audio out.

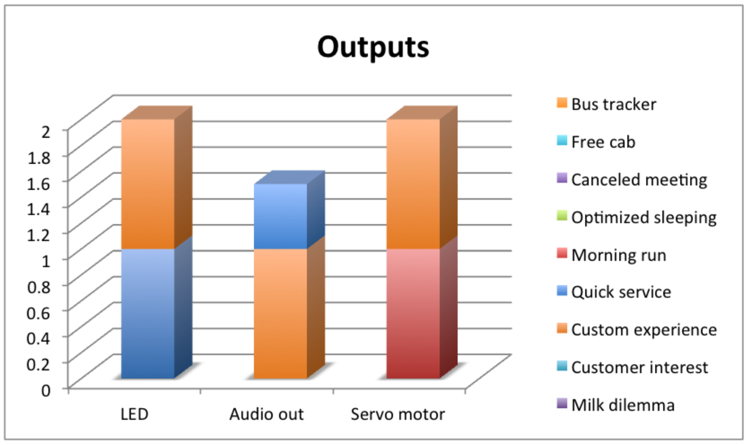


Table 3.3: Outputs

3.6 Sensor dependencies

To get valuable information about which sensors are the most important and most useful we tried to analyze the scenarios when only taking into account for one solution and making it the top in the hierarchy. Some scenarios have more than one interaction with sensors and since they are not related they were treated as “separated” scenarios when analyzing them. We also separated the trigger listener solutions from the sharp trigger ones. To get any information out of it we listed the sensor with one point at the top and below it the 0.5 solutions. If there were more than one with 1 point the sensor were listed 2nd and the 0.5 sensor as 3rd. Which sensor were at the top were determined by its score in table 3.1-3.3. But if a solution were equally good a note was made about this to handle it later. For example if we take a scenario like the Forgotten card would look something like this.

Title: Forgotten card
When you closed the door behind you your house sensed that you were the last person to leave
1. iBeacon
2. Motion detect
3. RFID, Button

and locks the door together with alarming the first floor.
1. Servo motor

It also sense that you forgot to pack the company card you need to enter your office and sends a warning to your smartwatch.
1. iBeacon
2. RFID

To make it easier we color-coded them where yellow is output, green listeners and blue sharp triggers.

After going through all the scenarios three trees were built which represent one for each sensor group. The arrows points towards sensors that it can replace in the scenarios we have listed. This also means that they can be used in similar ways.

3.6.1 Input trigger listeners

As Input trigger listeners we found a dependency tree suited for object tracking. iBeacons, Camera and RFID can all be effective for this task but works somewhat differently. iBeacons are optimized for bigger objects like humans, possible tracking who and how many people that are contained in a room but needs to have a device that detects the iBeacon. A camera could do the same provided that it includes human recognition algorithms that might be unnecessarily advanced for the intention of prototyping. RFID is more or less an iBeacon but optimized for shorter ranges.

Audio in is often used for a scenario specific to voice recording, GPS for outdoors location tracking. These sensors are therefore independent and have no obvious dependencies. Force sensor is independent as well due to the ability to measure weight. In the Quick service scenario, when measuring warm beverages it can be replaced by both Temperature and Gyroscope.

In our scenarios Gyroscope and Accelerometers as listeners are usually used for the same purpose which is that they track movements over time.

Temperature, Barometer and Humidity are often used as sensors listening for air changes. For prototyping we found that these sensors are often used to track sudden or over time changes. In these cases the Temperature sensor can replace the others. In a scenario where you take weather into account Barometer and Humidity sensors becomes essential.

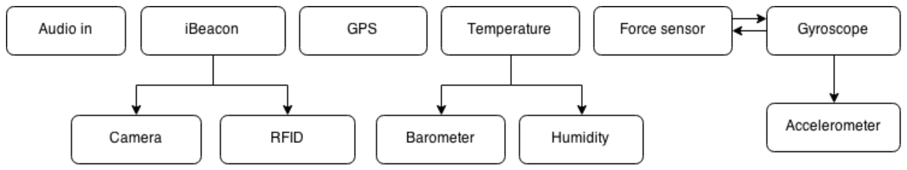


Figure 3.2: Trigger listener dependencies.

3.6.2 Input sharp triggers

The behavior of Input sharp triggers tends to act in a similar way. The main difference in their functionality is identification of where the triggering source comes from. In the bus tracker scenario we used sharp triggers to know how many people that are currently on the bus. In this case we do not care about who these persons are and can therefore prototype the scenario with a simple motion detector, increasing a counter by one when a passenger enters and decreasing by one when a passenger left. The responsive lights scenario is instead identification vital, as the light behavior is customized to the needs a specific person. Here it is of more importance if the one triggering is you or perhaps your dog.

As sharp triggers we found that a Gyroscope and Accelerometer always tend to do the same thing. As sharp triggers it does not matter which way something is rotated, only if it is moved. Therefor the difference in functionality becomes irrelevant for most cases.

The way Force sensor and Light sensors are used can be compared to a normal light switch that is either on or off. The Force sensor do not care about how much force it receives, only that it is either being pushed or not pushed. In the same way we used the Light sensor a trigger for either dark if it is covered, or bright if it is not covered.

In figure 3.3 iBeacon, Motion detector, RFID and Button can all work as area triggers. iBeacon, compared to a Motion detector, sense which specific person that enters an area but requires that they wear a smart device. As area triggers, RFID and Buttons can be used but would require the user in some way to physically trigger something.

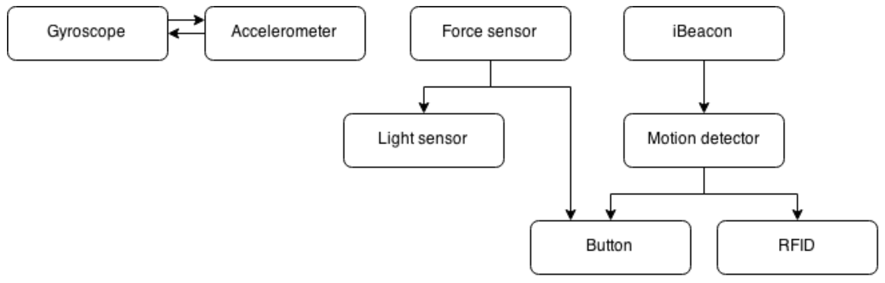


Figure 3.3: Sharp trigger dependencies.

3.6.3 Outputs

The output sensors used for these scenarios cannot be replaced by one another and therefore have no dependencies.

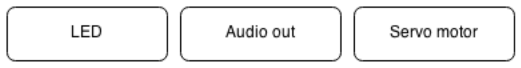


Figure 3.4: Output dependencies.

3.7 Conclusions of sensors for prototyping

These graphs and tables are based on our scenarios that we have listed in the beginning of the chapters if other scenarios had been made different results are a possibility. To get a more statistical strength to it a lot more scenarios would have to be listed.

From these graphs and trees we can determine which of the sensors that are more useful and might be better to implement support for in a prototyping tool. For both input categories you can see that iBeacon is a good sensor to work with. It is used in most scenarios that are listed and from the trees they are at the top of it on both of them. The downside is that they need an external device to detect the iBeacon and do something with that information. Since gyroscope and accelerometer have about the same usability in these scenarios support for one of them might be enough as a beginning. Force sensor is also a good sensor that has a wide variety of usage but it works best as a sharp trigger since an accelerometer or gyroscope could be sufficient in the listening category. Temperature sensor might also be a good one to start with since it can replace humidity and barometric sensors in our scenarios. As for the ones without any dependencies are not as important and more specific. They should definitely be supported but, not as main focus.

For a finished prototyping tool, all or most of the sensors should be supported. However, there has to be a priority of which sensors that are most of value to start with.

4 Exploration - Prototyping methods & tools

This chapter introduces methods of how to program using visual programming environments. There will also be a review of existing prototyping tools followed by a tool comparison to gain an understanding of which audience the tools are adapted for. This chapter was restricted to software prototyping tools and other ways of prototyping were not taken into account. This was due to time limitations and we felt that this was the direction of the project we wanted to go.

Introduction to prototyping

A prototype is a powerful tool to get an early representation of what finished product might look like. The prototype can be used to get realistic reactions when testing a user experience, or to convince your boss that your idea is something to run with. The prototype can be represented in many different ways, which also means that it will be represented in many different levels of realism towards a complete product.

When bigger companies start a new project they normally have a vague idea of what they want as a finished product. A common way to realize this project plan is to do a strict list of requirements that the product must conform to, then implement functionality around these requirements and finally test, if the product works on the market. This method is known as waterfall model [23]. The problems with this method may vary. The project team might find the product to function and sell exactly as planned. But if it does not, the project suddenly requires a surprisingly bigger wallet.

By introducing prototypes in an early state, even before the creation of Quality Requirements, the company can avoid many problems that would occur at a later stage.

Prototyping is often done agile with rapid iterations. The prototyping iterations can be used with the PDCA cycle [4] to get fast results on the prototype. These cycles can be very short compared to a full scale process. E.g. if you only change a very small feature in the prototype and then test it to see if it was better or worse than before.

4.1 Visual Programming Environment

There are many parts of programming which makes it complex to learn [24]. A program consists of a number of instructions needed to perform a specific task. These instructions can be written in different formally constructed programming languages which all can include different syntax. The syntax of a language includes the set of rules defining which combinations of symbols are considered correct. Knowing the syntax by heart might be essential for effective programming. This means that even experienced programmers encounter large thresholds when switching language, even though the logic flow of their program is similar or even the same.

With VPE it retains the programming concept of giving instructions but lift out the problem of syntax that normally steal focus from the logical flow of the system. In prototyping, this decreases the learning curve of creating an actual interactable prototype and makes the iteration time for inexperienced programmers faster.

Michael Winberg, co-creator of the flow-based inspired prototyping tool Noodl states that the iteration time for an experienced programmer that knows the used language by heart is about the same, or even slightly slower. For him the real value lies in the ability to be able to demonstrate the logical structure of an prototype to customers and team members, which he experiences a significantly better and more impressed response towards showcasing a JavaScript library [25].

4.1.1 Flow based VPE

Flow based programming is a way of creating powerful prototypes that are becoming increasingly popular. These tools are based on a library of visually displayed components which can consist of an activity linked to either an input or output to create logical flows of different interactions. Hidden in the components are structures of code which the user do not have to be exposed to, which makes it easy for novice programmers to work with. The components are connected by arrows representing the way in which the data is flowing [24].

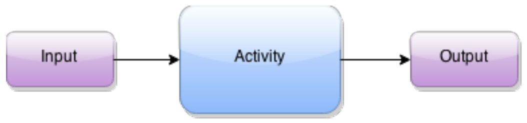


Figure 4.1: Basic node components

In the subject of Flow-based prototype you might encounter the question of Time-line based prototyping tools and why it might not be suited for this type of interactable prototypes. Paul Colton, founder of the prototyping tool Pixate states that:

"Time, at least for interaction, implies you know the exact flow that your user's going to take. Interaction by definition means it's random, and I think that's fundamentally why timeline doesn't work. The users might reverse the order in which they move their finger and you're not going to know that ahead of time" [26].

However, when simulating a case where you know how the user is going to act a timeline will be very useful [27].

From the beginning of flow-based programming all the normal functionality of normal programming was there [28]. This is something that has been changed in recent tools that chose to keep logical programming things such as loops or if-statements away from their tools e.g. Noodl or Pixate. Some flow-based tools choose to keep the code completely away, which might create frustration for experienced programmers that suddenly are not able to do what they want. Other tools supply the prototyper with an option of altering components to fit their need that might not be optimal for novice programmers. Which way that is considered the best cannot be decided. This depends on which target group the tool is developed for.

4.1.2 Practical trigger-action

Imagine that you live in a fantastic world where almost every single object is connected. All the endless tailored experience solutions you could create. But are they really necessary?

Based on a study made by Blase Ur et al. [29], when asking everyday users of what possible new features and events they would want their smart things to do they are often quite small solutions. Most limits themselves to single triggers and single actions [29]. "When I wake up I want the coffee to be ready". "When I travel home from work I want my wife to be notified". These types of actions are called trigger-action programming and normally you use it as "if - trigger, then - action". These simple one trigger and one action works in 78% of the cases and the rest 22% required an additional trigger, an action or both.

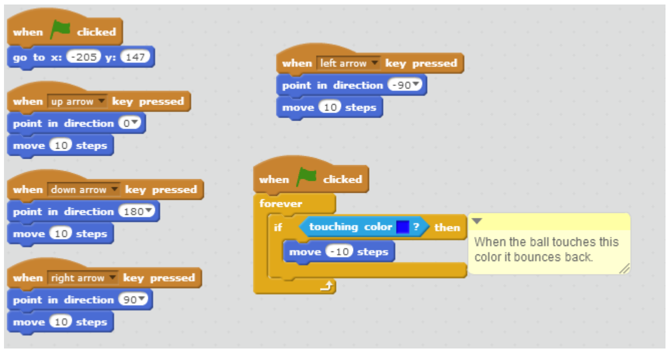
In practical trigger-action a combination of trigger-action is commonly called a recipe. In the study, Blase Ur et al. noted that over three-quarters of the participants agreed that "it was easy and intuitive to create recipes", that they would be "interested in creating recipes of this sort in daily life", and that they thought they "could handle a more complex programming interface". This concludes that opening up the complexity of programming would create conditions for new user experiences.

Figure 4.2: Example of a practical trigger-action recipe

4.1.3 Block programming

There are some variations to block programming but basically the main idea is that you have an object you want to manipulate in some way then you click on it and drag it to a workspace where you drop the object. An example is the MIT produced language Scratch that is a tool that lets the user use predesigned simplified programming functions. The main goal when creating Scratch was that it should be a tool for young people to learn programming, working in teams and share their work with others. Because of the nature of the drag and drop principle this is a good way to introduce people with no prior programming experience into learning how the operations work and how to think when building code [30].

What is common with all of the block programming languages is that they have a visual representation of the code and you get feedback when you run the program in form of something moving or playing some sound. There are two other tools similar to Scratch; Greenfoot and Alice, that are targeted at a bit older audience, they work in similar ways but are not only graphical representation of the codes like Scratch is. These programs lets the user understand the concept of object-oriented programming by introducing the class systems that most programming languages are based on [30].



```
when clicked
go to x: -205 y: 147

when up arrow key pressed
point in direction 0
move 10 steps

when down arrow key pressed
point in direction 180
move 10 steps

when right arrow key pressed
point in direction 90
move 10 steps

when left arrow key pressed
point in direction -90
move 10 steps

when clicked
forever
    if touching color ? then
        move -10 steps
```

When the ball touches this color it bounces back.

Figure 4.3: A simple example of a program in scratch

The main goal with block programming is that it should be very easy to use for everyone without any type of prior programming experience. It should be easy to create new projects and just jump into the programming without any long introduction. This requires the blocks to be pretty much self-explanatory. It is a great way to introduce young people to programming and letting them explore with it to create games and things they have on their mind [30].

4.2 VPE tools

This section will describe strengths and weaknesses of existing prototyping tools followed by a tool comparison to gain an understanding of which audience the tools are adapted for. Note that these types of tools are being rapidly updated. The following reviews are written at the date of this report. The tools' appearance and functionality might have changed by the time you read this report.

4.2.1 Noodl

Noodl is a tool that is inspired by Flow-based VPE and created by a design firm for Hi-Fi prototyping. The tool is structured on visual nodes running JavaScript and includes a wide range of programming logical nodes, which makes the tool flexible enough to create almost any interaction or visual animation. The greatest value of using Noodl for prototyping instead of coding is not only to create fast design iterations for novice programmers. The possibility of showing how that actual data flows through your prototype to a Non Programmer may be just as valuable.

Noodl is suited for medium scope systems with Hi-Fi results, meaning that it is great for creating prototypes that looks and feels exactly like the real product would do, but might get a bit too complex to build if you are to build a larger system.

A great feature of Noodl is that it is run through web and can be connected to through any device using a browser. This means that you can test and demonstrate an application in real time on your phone or tablet without having to compile anything in your tool that makes for fast design iterations.

The tutorials in Noodl makes the tool easy to understand and get into, but becomes quite complex as soon as you want to create something more advanced. Users with some basic programming knowledge will probably not have any trouble using it, but non programmers might find the tool difficult to use [31].

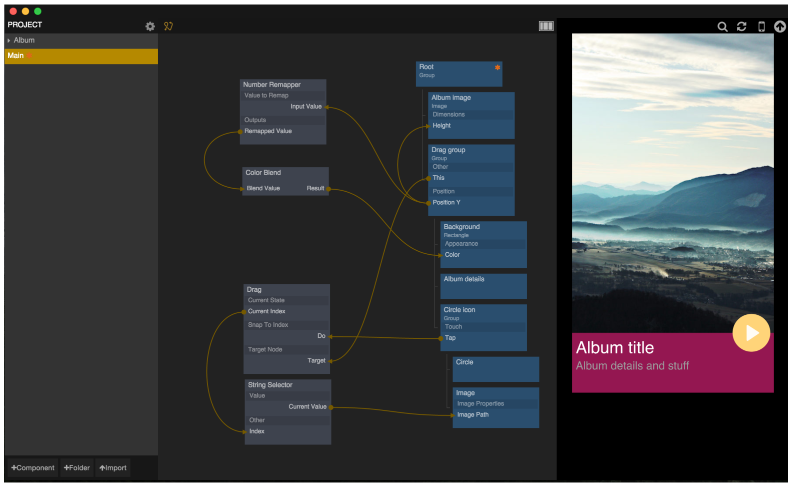


Figure 4.4: Screenshot of Noodl from one of its tutorials

4.2.2 Pixate

Pixate is a graphical layer based prototyping tool. Its focus is to help designers to create their prototypes with animations and interactions without knowing any programming. Their goal is to make fast representation of an application and then the ability to change it with fast iterations of the design. Pixate also lets you use a team function where your design team can make changes to the project and also lets stakeholders see the progress of the design. The workspace consists of a “screen” where you place layers, which is squares that you can interact with.

This system is best used for systems where the user has an application of some sort they want to try out. The screen space is quite limited which makes bigger systems almost impossible to create while still having control of what everything does. Thus this system is powerful when the user only wants to have for example 2-3 screens they want to prototype. The tool is entirely based on layers for visuals and all interactions for example a tap function is dragged onto the layer that gives it that property. This makes the tool quite powerful because it is easy to just start doing things without any real knowledge of the tool. The hard part, as in most tools, is how everything ties together and how the logic should work. As the system gets bigger it is almost impossible for someone else that has not built the system themselves to understand how things work.

To try out the prototype design the user has to download Pixates application and sync the accounts. However whenever you make a change in the project the prototype updates on the phone almost simultaneous. Overall it is a very powerful tool that is simple to use and lets the user do really realistic prototypes [32].

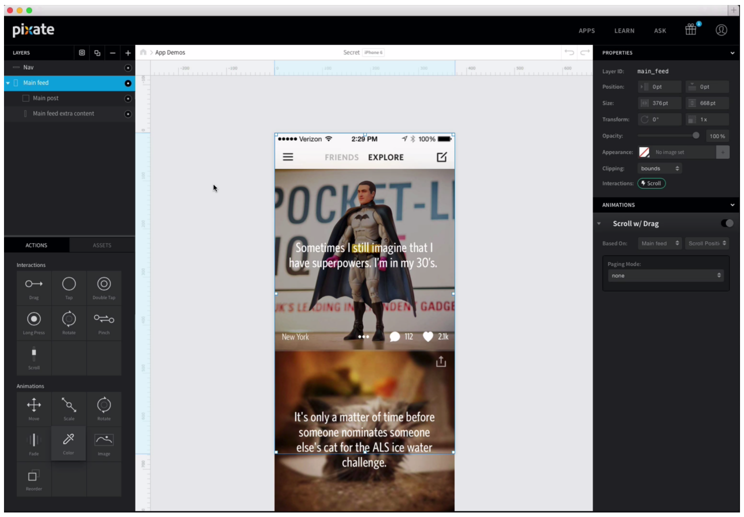

pixate
APPS
LEARN
ASK
LAYERS
Nav
Main feed
Main post
Main feed extra content
App Demos
PROPERTIES
ANIMATIONS
Scroll w/ Drag
ACTIONS
ASSETS
Interactions
Animations
FRIENDS
EXPLORE
Sometimes I still imagine that I have superpowers. I'm in my 30's.
New York
It's only a matter of time before someone nominates someone else's cat for the ALS ice water challenge.

Figure 4.5: Screenshot of Pixate from one of their video tutorials [33]

4.2.3 Invision

Invision is a tool that lets user illustrate their design ideas with various interactions in a fast and quick way. To create this, Invision consists of an overview that shows you all the screens you have added to the application and if you click on these screens the user comes to an “edit view”. This view consists of four different modes; preview mode, build mode, comment mode and history mode, which are quite self-explanatory. How the user creates their design prototype is simply by importing pictures into the project and in the build mode arrange them as they see fit. To make the pictures interactable the user hotspot an area of the picture and enter what the user want to happen to this area. However these interactions are quite limited, all they can do is transitions to other pictures by either tapping or sliding on the hotspot area. To create animations you have to upload gif-files that make to animation you want and make them trigger when you click on a hotspot.

This tool shines when the designer want to create a fast prototype to show a simple overview of how the looks of your design will look like. A downside is that you have to create the pictures in other tools and then import them, which requires the user to be somewhat experienced in picture editing. In the tool there exists a very good team system which lets different persons change in the project at the same time and also comment on changes or pictures in the project. It also has a version control that lets the users see different versions of the design and revert to previous designs easy. To try the prototype you can share the project by almost all means (URL, SMS, email) which simply opens the prototype in a web browser where you can try it out [34].

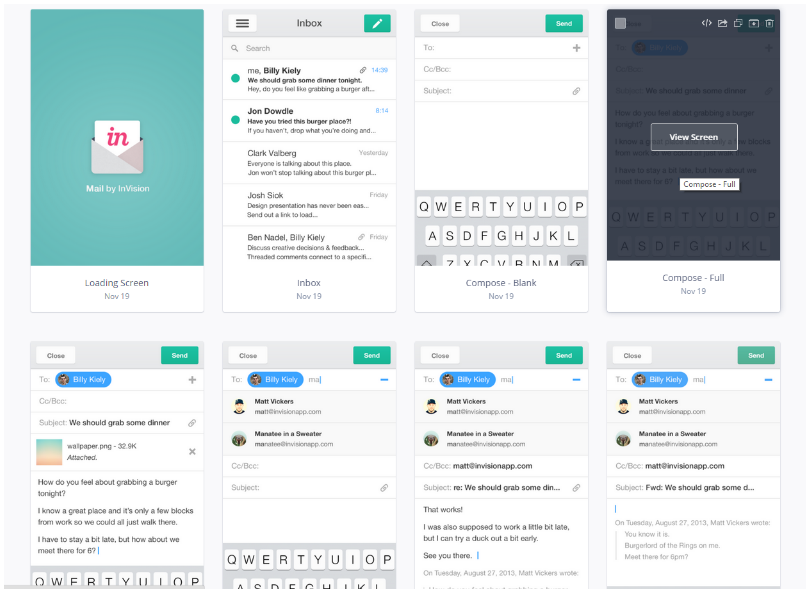


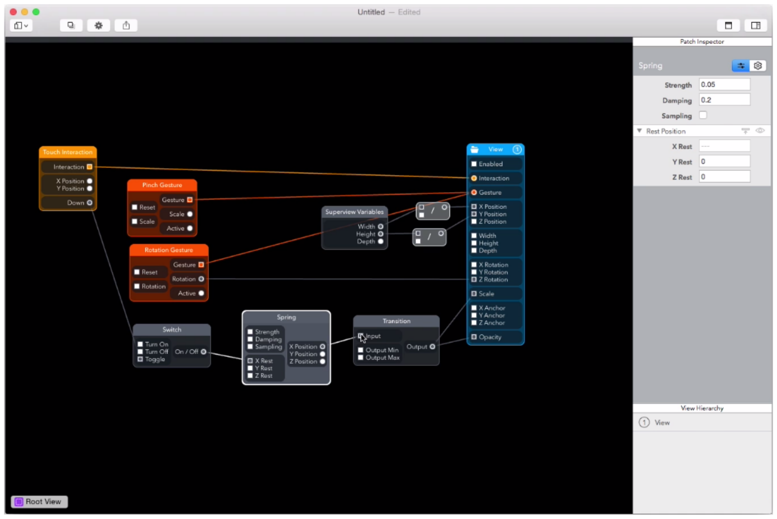
Figure 4.6: Screenshot of Invision from one of their premade illustration projects

4.2.4 Form

Form is a relatively new prototyping tool that is created by RelativeView but got bought by Google. The tool is a flow-based system where you create nodes (or patches as they are called in form) that you interact with other nodes by connecting them with each other. Form is a versatile tool that lets the user modify the nodes on a very detailed level. The whole window consists of a big workspace where you can add you nodes. When you select a node an option menu will appear on the right side of the window where you can edit the node. This tool is specialized on systems in mid-scale that is one or a few screens or a feature interaction. Even though bigger more complex systems are possible this will be overwhelming and probably better to show this part of the system in another prototype.

Form only works for iOS and not for the web browser, rather it uses an application in the users phone to try out the prototype. This works pretty much the same way as Pixate, when you update something on your computer the prototype in the phone updates. Since the prototype runs on phone it gets access to the phones sensors and various functions, this allow the user to interact with E.g. the camera view. It also has a good feature that lets the user group together nodes, this makes the workspace cleaner and you do not have every node you are using visible.

Form is a moderately hard prototyping tool with some tutorials to help the users get going. However when you get to know the system it is very powerful and you can build almost all kinds of systems within the limits of the nodes [35].



Patch Inspector

Spring

Strength	0.05
Damping	0.2
Sampling	☐

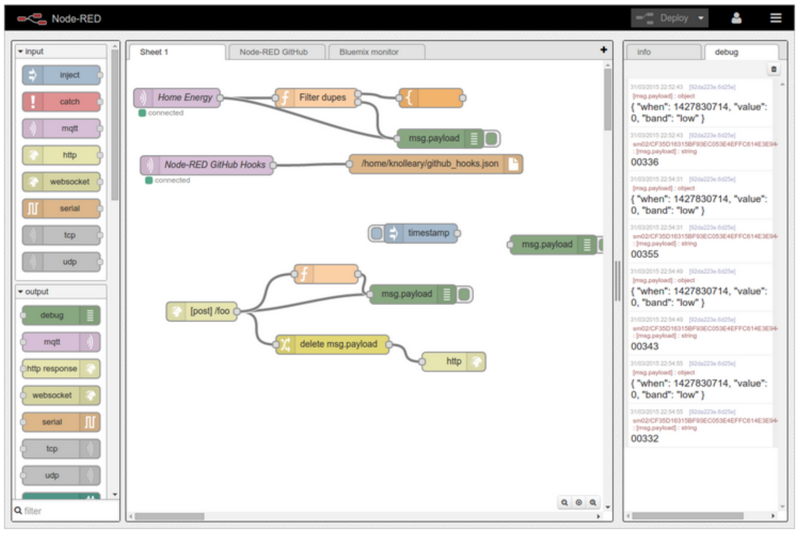
Rest Position

X Rest	---
Y Rest	0
Z Rest	0

Figure 4.7: Screenshot of Form from one of their tutorials [36]

4.2.5 Node-RED

This is a tool customized for actual sensor integration and proclaims itself as a visual tool for wiring the Internet of Things. In the workspace the user build a visual flow of nodes in which they use a clear distinction between different types of nodes and which particular functionality they are used for. However, the level of technical difficulty is high, considering they use terminology and nodes suited to experienced programmers which for a Non Programmer, or even limited programmers, is hard to understand. Just setting up the tool to work takes a while to understand and to get it correctly done is hard. The tool is suited for connecting hardware to APIs and a variety of online services and is also completely open source [37].



```
31/03/2015 22:52:43  [92da223e.6d25e]
[msg.payload] : object
{ "when": 1427830714, "value":
0, "band": "low" }
31/03/2015 22:52:43  [92da223e.6d25e]
sm02/CF35D16315BF93EC053E4EFFC614E3E94
: [msg.payload] : string
00336
31/03/2015 22:54:31  [92da223e.6d25e]
[msg.payload] : object
{ "when": 1427830714, "value":
0, "band": "low" }
31/03/2015 22:54:31  [92da223e.6d25e]
sm02/CF35D16315BF93EC053E4EFFC614E3E94
: [msg.payload] : string
00355
31/03/2015 22:54:49  [92da223e.6d25e]
[msg.payload] : object
{ "when": 1427830714, "value":
0, "band": "low" }
31/03/2015 22:54:49  [92da223e.6d25e]
sm02/CF35D16315BF93EC053E4EFFC614E3E94
: [msg.payload] : string
00343
31/03/2015 22:54:55  [92da223e.6d25e]
[msg.payload] : object
{ "when": 1427830714, "value":
0, "band": "low" }
31/03/2015 22:54:55  [92da223e.6d25e]
sm02/CF35D16315BF93EC053E4EFFC614E3E94
: [msg.payload] : string
00332
```

Figure 4.8: Screenshot of Node-RED from one of their tutorials [38]

4.2.6 If this then that

The application is based on practical trigger-action and lets the user create small but effective applications. IFTTTs main philosophy is that everyone can make applications that can be functional and useful in a very short time. IFTTT uses other application and connect them with some functionality that the user want to use it for. For example if someone posts something you are interested in on twitter you get a SMS saying that they have tweeted something [39].

IFTTT have been separated into two different categories where they call one IF and the other DO. The main differences between these are that in the DO recipes there exists an application that IFTTT have done that are connected to hardware e.g. Do camera. The DO camera application lets the user create recipes about what will happen when pictures are taken. IFTTT is a very easy to use application with basically no learning curve at all. More or less anyone can try it out and get working applications to use. A negative aspect is that if the user wants to use a channel or an action that is not supported then it is impossible but the developing team is working to get more and more channels and actions in all the time. Another is that it does not support more conditions e.g. you want to have two triggers, which is not possible. However the easy to use recipe creating and the easy to both share your recipes and take part of others recipes makes IFTTT a very powerful application to make easy applications [39].

Figure 4.9: Screenshot of IFTTT from a recipe

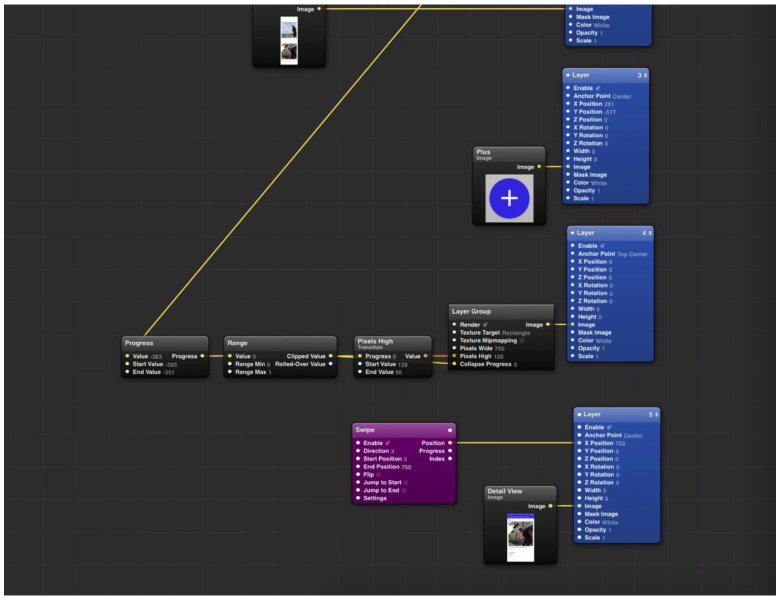
4.2.7 Quartz Composer with Origami

Quartz composer (QC) is a flow based graphical rendering tool which is developed for OSX. The original version of QC is not suited for prototyping because it does not allow interactions with phones and is more of a rendering program, but with origami this became a real thing. Origami is created by Facebook and with this, new nodes (called patches) were added into QC that made interactions with phones viable [40]. Since QC was not intended to be used as a prototyping tool it is not running as smooth as it could and it crashes on occasions. QC is similar to Form and Noodl and might even be an inspiration to both of them with the visual programming, however not as good suited for prototyping [41].

Two strengths with QC are that you can easily comment the nodes by marking an area in the workspace and then write what this segment does. The other is that it has programmable nodes were you can own code to work with shaders or if you want to use JavaScript [41].

QC is suited for small systems with one screen and some interactable feature to it. As in all of the systems users can build full working systems but it becomes complex fast and almost impossible to keep track of it. However, here is where the commenting functionality shines that lets the user, and persons who have not built the system, get a more clear explanation of the system.

QC with its visual programming is not that hard to learn as a new user. The main complexity is the lack of good documentation, which can make users frustrated. Because of this the users more or less needs to watch and do tutorials to get a good understanding of how the tool works.



Image

Image
Mask Image
Color White
Opacity 1
Scale 1

Layer 3

Enable ✓
Anchor Point Center
X Position 281
Y Position -577
Z Position 0
X Rotation 0
Y Rotation 0
Z Rotation 0
Width 0
Height 0
Image
Mask Image
Color White
Opacity 1
Scale 1

Plus
Image
Image

Layer 4

Enable ✓
Anchor Point Top Center
X Position 0
Y Position 0
Z Position 0
X Rotation 0
Y Rotation 0
Z Rotation 0
Width 0
Height 0
Image
Mask Image
Color White
Opacity 1
Scale 1

Layer Group

Render ✓ Image
Texture Target Rectangle
Texture Mipmapping
Pixels Wide 750
Pixels High 128
Collapse Progress 0

Progress

Value -383 Progress
Start Value -383
End Value -351

Range

Value 0 Clipped Value
Range Min 0 Rolled-Over Value
Range Max 1

Pixels High
Transition

Progress 0 Value
Start Value 128
End Value 96

Swipe

Enable ✓ Position
Direction X Progress
Start Position 0 Index
End Position 750
Flip
Jump to Start
Jump to End
Settings

Layer 5

Enable ✓
Anchor Point Center
X Position 752
Y Position 0
Z Position 0
X Rotation 0
Y Rotation 0
Z Rotation 0
Width 0
Height 0
Image
Mask Image
Color White
Opacity 1
Scale 1

Detail View
Image
Image

Figure 4.10: Screenshot of Origami from one of their tutorials [42]

4.2.8 Scratch

Scratch is mostly based on graphical animation, or they represent the code with animation that gives you a visual representation. For example you want to do something ten times then you drag out a repeat icon and into that you add what you want to happen and when it should happen (see figure 4.11). This eliminates the syntax errors that are shown in traditional programming as you build your program as LEGO some parts fit with others but other do not. However when you make an error this is more an error in the way you think the program works rather than not using the program in the appropriate way [30].

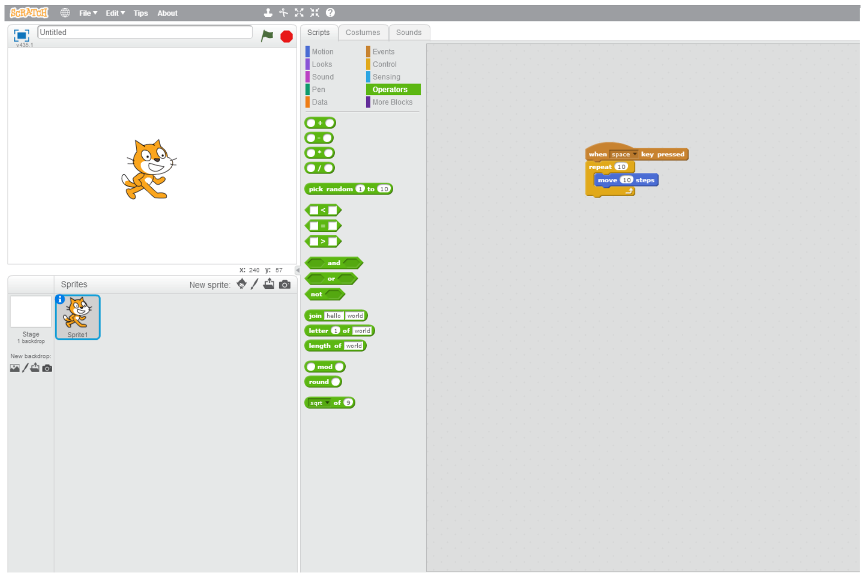


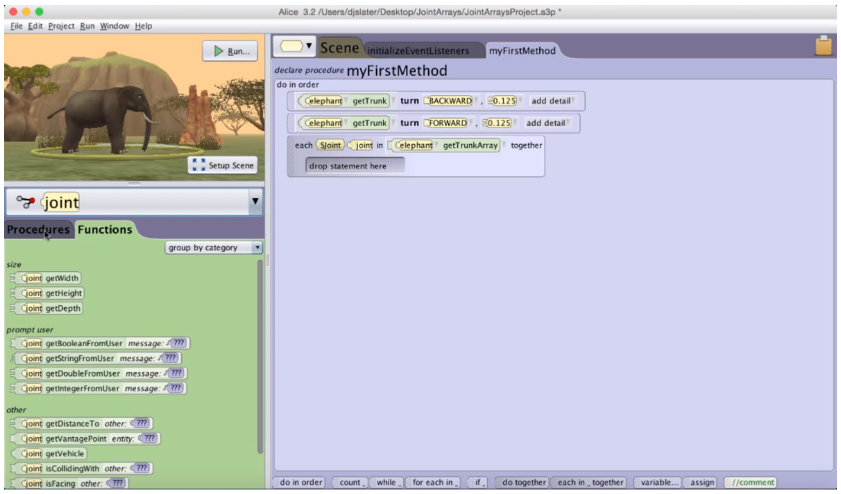
Figure 4.11: Screenshot of Scratch when creating a simple program [43].

4.2.9 Alice

Alice is a programming environment that is focused on persons with no prior programming knowledge. Alice is based on block-based programming where the users drag and drop blocks that represent programming code. The environment it set as a 3D game creator where the user is encouraged to use code blocks to interact with the 3D objects that have been created [44].

Alice has removed the possibility of having syntax errors, which is a frustrating feature for new programmers. Instead the errors they receive is only that of logical mistakes which will be shown visually as the objects will not behave the way the programmer intended it to be [44].

Alice block programming is easy to use and in the same time it is educational for people that are learning the logical thinking of programming, which is the main goal of Alice. It has been shown that persons with no or limited programming skills acquired higher grades on an introducing course in programming when they had been working with Alice before than persons who had not. They were also more likely to continue on a second programming course after it [45].



```
declare procedure myFirstMethod
do in order
    elephant getTrunk turn BACKWARD , 0.125 add detail
    elephant getTrunk turn FORWARD , 0.125 add detail
    each SJoint joint in elephant getTrunkArray together
        drop statement here
```

```
size
joint getWidth
joint getHeight
joint getDepth

prompt user
joint getBooleanFromUser message: ???
joint getStringFromUser message: ???
joint getDoubleFromUser message: ???
joint getIntegerFromUser message: ???

other
joint getDistanceTo other: ???
joint getVantagePoint entity: ???
joint getVehicle
joint isCollidingWith other: ???
joint isFacing other: ???
```

Figure 4.12: Screenshot of Alice from a video tutorial on their website [46].

4.3 Tool comparison

In figure 4.13 we have made a graph which grades the tools by two criteria, which level of complex system it can build and how easy it is to use for non programmers. With complex systems we include how big systems it can make and also how complex the systems can be on a logical level. To determine how hard they would be for a Non Programmers we had to try set ourselves in the mindset of a non-programmer and what problems they would have with the tools. This graph is based on our own experiences when trying the tools and what we have read about them.

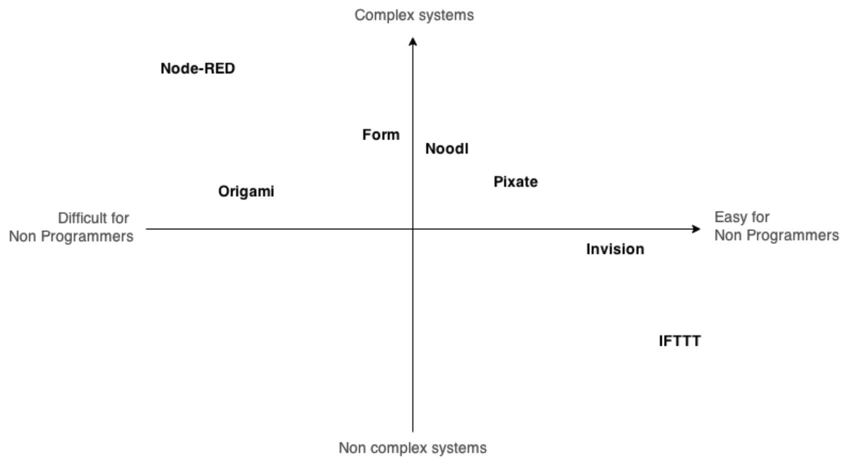


Figure 4.13: Comparison between the tools (Alice and Scratch not included because they are not really made for prototyping).

5 Exploration - Sensor-to-tool integration

This chapter will give a brief explanation of how different wireless sensor communication technologies work. It will also include various sensor products that simplify the usage of sensors for prototyping. This will be culminated to an example where we try to integrate sensors into one of the explored tools in chapter 4. The conclusions of this chapter consist a list of valuable tool features used in our tool design.

5.1 Wireless sensor communication technology

This section will give a short introduction to the most common wireless communication systems that are suited for sensors.

5.1.1 Bluetooth

Bluetooth is the most commonly known technology for connecting devices wireless. The technology is based on radio transmission called Frequency-hopping spread spectrum. On a very basic level the devices jump around on different transmission frequencies that are predetermined by the "main device". The data itself is packaged into packages and sent on one of the 80 channels (starts at 2402 MHz and each channel is 1 MHz). Hopping around these channels are done 1600 times each second. On the newer versions of Bluetooth there are only 40 channels (each channel are 2 MHz) [47].

Bluetooth works on much shorter distances than other radio technologies (up to 100 meters). Bluetooth 4.0 or Bluetooth Smart as it is called is a newer version of Bluetooth that drains the battery very slowly and for sensors this is very good so you do not have to change batteries all the time [47].

5.1.2 ZigBee

The name ZigBee come from the nature and perhaps as expected the behavior of bees. It was first noted by the 1960's Nobel Prize-winner Karl von Frisch that the bees, when looking for new places to gather food sends out a worker bee to locate suitable areas. When the area is located, the bee travels back to the hive and perform a waggle-dance, communication the distance, direction and type of food. The distance however is not perfect. When they arrive to the location there could be a 5-6 meter error, which is why the bees start zigging and zagging around to find the correct location [48].

Zigbee utilizes a mesh network structure for sending and receiving information. A mesh structure is built by that every unit in the system are connected to each other in some way, either direct or indirect and there are normally more than one way to reach a destination. This is commonly used in routing when transmitting information from a network to another over long distances. The advantages with this type of topology are that it is reliable, low cost and covers large areas. In Zigbee this is used to transmit information to nodes that might not be in reach of the main station of the unit that send the information. Because of this it is suitable for IoT devices and large industries [49].

5.1.3 Alljoyn

Alljoyn is a technology that connects devices and lets them communicate with each other. It differs from both Bluetooth and Zigbee since it is not a wireless communication protocol. It is more of a standard of how to make wireless communication proximity based. In standard Bluetooth you connect two devices with each other in a peer-to-peer connection. With an Alljoyn implementation two devices that does not have the range of each other can send information through the Bluetooth device that are in range of the other one. The main advantage of this is that you do not need a central main station to communicate with the devices. The idea of Alljoyn is that each device acts as a provider that tells the other devices which services it can provide at the same time it also acts as a consumer and tries to take advantage of the services the other devices in its vicinity. The device can have a specific role as well like consumer if the user want to [50].

Alljoyn framework is open source, which makes the system more alive, and creates a community around it. This will make it easier for companies that what to tailor their product to an Alljoyn system. It also supports multiple programming language(C, C++, Java, C#, JavaScript and Objective C) which makes it even more versatile [50].

5.2 Sensor products

These sensors have been put together in different kits that have its main goal to simplify the usage of sensors for people of different skill base.

Arduino

Arduino is both hardware and software that lets users interact with components beyond the computer. The hardware consists of microcontroller boards, which lets the user insert various components into its inputs or outputs. By using Arduinos' software program the users can modify these boards and interact the components. Everything is open-source which lets the users use the code in other programs of their choice. Arduino programming environment is relative easy and based on processing it also works on most platforms which many of the similar systems do not support [10].

Raspberry Pi

Raspberry Pi is a small computer the size of a normal phone. Its main purpose is to help young people get interested in programming and computing. Raspberry

Pis' functionalities are almost the same as a normal computer. You can plug in a keyboard, mouse, monitor and even an Ethernet cable and interact with its operative system. Raspberry Pis' goal was that everyone could have a computer and since it is not that expensive encourage kids to play with it and try new things without the fear of destroying it [11].

Texas Instruments sensor Tag

This is s sensor tag that includes ten different types of sensors inside of it with support to add more. The sensors that are embedded in the kit are light, digital microphone, magnetic sensor, humidity, pressure, accelerometer, gyroscope, magnetometer, object temperature, and ambient temperature. The device can be connected to the cloud using a Bluetooth to connect to a device that has the application installed on it. If you want to use Zigbee instead it has support for this that can be added to the sensor tag which makes it suitable for IoT experiences [51].

Estimote

Estimotes are sensor kits that consist of three types of sensors, an iBeacon, a thermometer and an accelerometer. The Estimotes exists in two variants, the stone and the sticker. The stone is a bigger sensor that has a better battery and signal range compared to the sticker that is smaller but weaker. To use the Estimotes the users need to know how to program to try out and build own applications for the Estimotes. To modify the Estimotes there is an API with some tutorials to get new users going. There exists both Android and iOS SDKs that enables users to use the code in any way they want and work in whatever program they want [52].

Smartphone

A normal smartphone today has various sensors embedded into it. These can be utilized when prototyping and you do not want to buy sensors to try out your systems. Most of the phones have support for applications that users can program themselves. Which sensors and how accessible they are differs from phone to phone.

5.3 Prototyping tool connectivity test

To get a deeper understanding of the complexity with sensor integration prototyping we decided to pick a scenario from chapter 3 and realize the experience with a prototyping tool. This gave us a deeper knowledge of Flow-based programming and also the work it would take to create a tool to support all possible sensors.

5.3.1 Tool of choice

The tool we are designing will be adapted for Hi-Fi results for medium scope systems. This means that the tool to create our prototype on does not have to be optimal for large scope systems i.e. complete application, but to specific functionality in a user experience. We also needed a prototyping tool that might not be built for sensor prototyping, but will at least be able to support it.

For our example we chose Noodl as the prototyping tool to prototype on. One reason behind this choice is the fact that it is the only Hi-Fi tool we have been researching that supports sensor integration as well as a visual representation of the application. Noodl is also the tool developed by Topp, which means that we could get direct support if encountering any problem.

5.3.2 Scenario of choice

Custom experience
You and your friend decide on taking a trip to the museum located in an old castle in your city. Neither of you have been there before. You do not really care for history and information of the artifacts, but enjoy the sentimental atmosphere this old castle has. Your friend on the contrary is a history junkie and reads every word of information that could be found. When you arrive to pay for the entrance you are both given an application in which you can state which type of experience you want to experience. You enjoy sound effects vivid stories so you chose to enter in horror mode, and your friend that does not care for that as much and want straight facts chose informative mode. When you and your friend start the route, the application senses in which room you are and subsequently presents different stories, information and various events depending on where you are and which artifacts you are currently watching.

For this test we figured that the above scenario fits our interests. From the sensor-to-scenario analysis we found that iBeacons is a suitable way to solve many prototyping problems and a natural sensor so further explore. Custom experience is a great scenario to do this.

5.3.3 Exploration-prototype design

When you arrive at the museum you will be given a smartdevice to use during the stay. On the main screen (figure 5.1) you will find a variant of experience options the museum are giving out. These experiences might be permanent or temporary depending on which exhibitions the museum are giving at the time. To select an option you slide the experience of choice to the right.

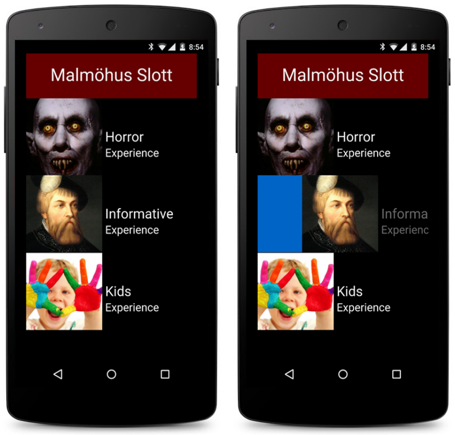
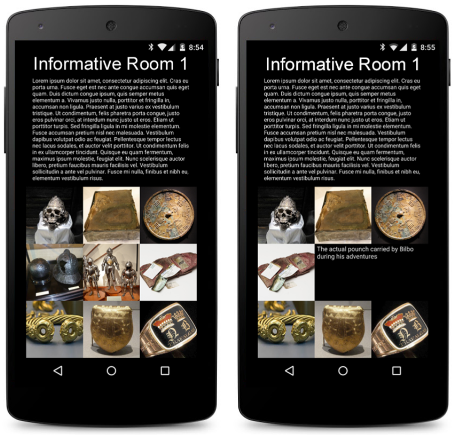


Figure 5.1: Hi-fi concept of main screen.

The informative experience is adapted to visitors looking for fun and interesting facts about the exhibition. The structure of the screens showed for different rooms can be different. In figure 5.2 you can see how the first room for Informative experience are structured. It contains an introducing text describing the room followed by pictures of every object in the room you are inside. To get more detailed information about a specific object you press its representing image.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras eu porta urna. Fusce eget est nec ante congue accumsan quis eget quam. Duis dictum congue ipsum, quis semper metus elementum a. Vivamus justo nulla, porttitor et fringilla in, accumsan non ligula. Praesent at justo varius ex vestibulum tristique. Ut condimentum, felis pharetra porta congue, justo eros pulvinar orci, at interdum nunc justo ut eros. Etiam ut porttitor turpis. Sed fringilla ligula in mi molestie elementum. Fusce accumsan pretium nisl nec malesuada. Vestibulum dapibus volutpat odio ac feugiat. Pellentesque tempor lectus nec lacus sodales, et auctor velit porttitor. Ut condimentum felis in ex ullamcorper tincidunt. Quisque eu quam fermentum, maximus ipsum molestie, feugiat elit. Nunc scelerisque auctor libero, pretium faucibus mauris facilisis vel. Vestibulum sollicitudin a ante vel pulvinar. Fusce mi nulla, finibus et nibh eu, elementum vestibulum risus.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras eu porta urna. Fusce eget est nec ante congue accumsan quis eget quam. Duis dictum congue ipsum, quis semper metus elementum a. Vivamus justo nulla, porttitor et fringilla in, accumsan non ligula. Praesent at justo varius ex vestibulum tristique. Ut condimentum, felis pharetra porta congue, justo eros pulvinar orci, at interdum nunc justo ut eros. Etiam ut porttitor turpis. Sed fringilla ligula in mi molestie elementum. Fusce accumsan pretium nisl nec malesuada. Vestibulum dapibus volutpat odio ac feugiat. Pellentesque tempor lectus nec lacus sodales, et auctor velit porttitor. Ut condimentum felis in ex ullamcorper tincidunt. Quisque eu quam fermentum, maximus ipsum molestie, feugiat elit. Nunc scelerisque auctor libero, pretium faucibus mauris facilisis vel. Vestibulum sollicitudin a ante vel pulvinar. Fusce mi nulla, finibus et nibh eu, elementum vestibulum risus.

The actual pounch carried by Bilbo during his adventures

Figure 5.2: Hi-fi concept screen of the first room.

The purpose of the prototype design is to prove the concept of user experience with iBeacon interaction. Our prototype therefore only contains an example for two rooms, the second room being a blank black screen.

5.3.4 Custom iBeacon detector for Noodl

The real challenge for this scenario is that we needed to send information from the beacons to the phone and then handle this information and send it to Noodl, which is supposed to be shown on the phone screen (see figure 5.3). We found a library that contained already implemented classes of how to handle beacons and how to receive information from them through Bluetooth. Different beacons have different bytes of information that works as a header to know which type of beacon it is. In our example we worked with Estimote so we had to set that when the header of the Estimotes are sent that is considered as a beacon.

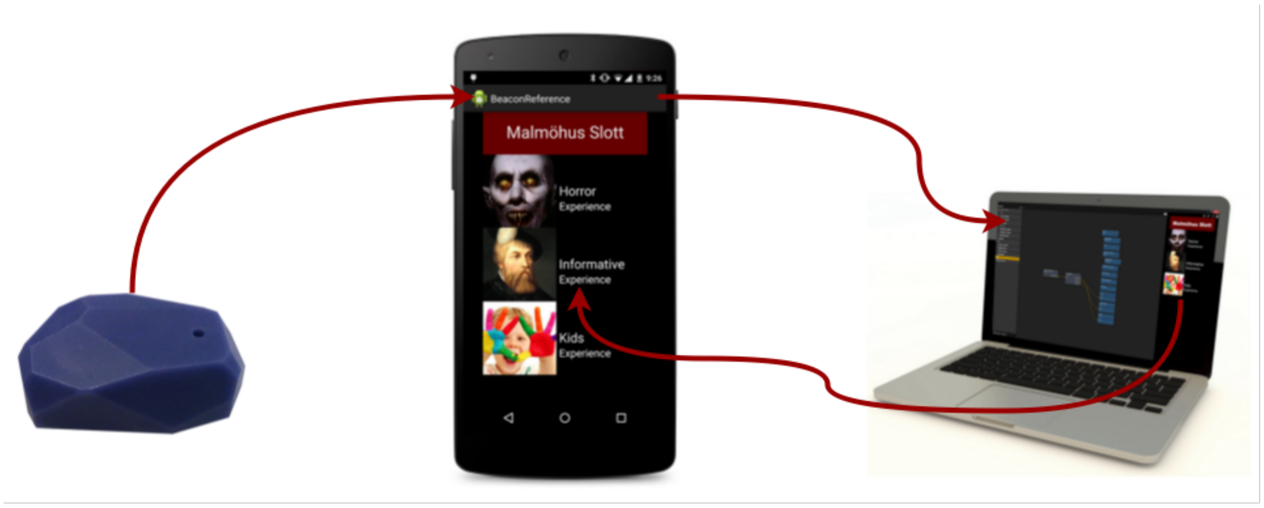


Figure 5.3: iBeacon application data flow.

To get the logic to work we had to make an android application that acts as a server that both receives information from the beacons and sends information to Noodl. The problem with that is that you cannot open a web browser to run Noodl and run the app in the background for sending information. The solution we came up with is that in the app use a WebView that more or less "streams" the web page onto the screen. The sending of information was done in services instead of activities because they do not need any GUI (since webview is used) and it had to be done all the time.

The data received from the beacon is excessive and need to be filtered. The only thing needed were the id of the beacon and how close it is to the phone. The distance is calculated by comparing signal strength of the beacon to a pre-determined list. The distance calculator is not very reliable, but can be used as "if you are close to the beacon". Because of this we set a limit that "if you are within one meter of the beacon send information about it". The information is sent through a websocket and it is basically a string in JSON format with the id of the beacon. This is sent to Noodl which shows different rooms depending on which beacon you are close to.

5.3.5 Noodl prototype workspace

Noodl works best if you use different kind of components. The components are basically your own nodes that users can create. With these, users can create generic framework that they can use in various ways depending on how the nodes were created. To get a more detailed description of how the Noodl flow will look like in the final prototype check Appendix A. But to get an idea of how complex the system would be you can see for example figure 5.4 which only shows one of the components needed for this system.

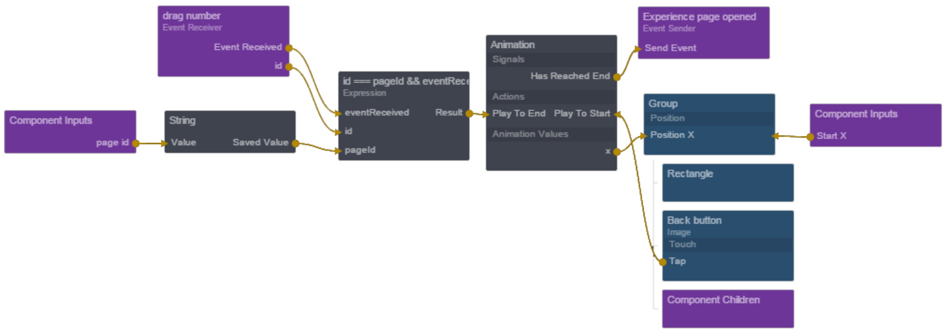


Figure 5.4: A component which handles the changing of experience pages.

5.4 Conclusions - Prototyping tool connectivity test

To build a prototype of this scale the complexity increases fast, which can be seen in Appendix A and figure 5.4. This is mainly due to dependencies between the different nodes, if this node is clicked something will happen but if another thing is clicked another thing would happen. And to abstract this into components will be hard for someone without this kind of logical thinking.

To integrate sensors into tools like Noodl is difficult and require experience in programming. There is currently no standard of how every sensor are to be integrated, which means that for a tool to work it has to implement sometimes completely different support for different kinds of sensors. E.g. when implementing support for Beacons, the tool requires different implementations depending on which kind of Beacon, if it is an Estimote, TI-SensorTag etc. For the tool to support all or most sensors on the market is therefore not feasible. When creating a tool suited for sensor integration one must decide on which sensor products to integrate. Due to time limitations, this is something we will not further discuss.

There are currently a few tools allowing sensor integration, but they all require a lot of coding from the user. Our tool needs to be built with sensor integration support as a main feature. If users are to code by themselves we believe that few will be bothered to try it out, thus losing out on a lot of potential users.

6 Valuable tool features

This chapter will present valuable features for a sensor based prototyping tool. The following features are based on features we found necessary, or missing from creating the sensor-integrated prototype in chapter 5. The features are inspired by existing tool features from exploration in Chapter 4. The second target of these features is to lower the tool learning curve and to make it as intuitive for Non Programmers as possible. Our ideal prototyping tool would be easy for both Experienced Programmers to not feel limited in creating whatever experience they want, as well as for Non Programmers to easily create small experiences.

Improved content navigation

Both Pixate and Noodl has a project content view in the upper left corner of the tool, listing layers (Pixate) or components (Noodl). In Pixate there is no sorting, but you can move and name the layers however you want. The workspace content of the layer at the top is placed above content of the layers below. Noodl uses a folder system. The structure of the names and order of folders and components is up to the user. To get a logical and intuitive structure of the project this requires the user to name and structure, which can be difficult when the projects grow large. In Pixate, where layers are used, users group them by dragging the layer in the list onto the layer you want to have as a parent. By doing this you can see which layer is used in other layers. This functionality is missing in Noodl where you have to enter the components to know which nodes are in them. Navigation becomes vital in particular with larger project if the user is not careful with naming the components.

We suggest functionality to give a clearer overview of which project components or layers the users are working on, and their relations to other project contents. Since both the folder system and the hierarchical system are good, we think of a solution to somehow switch between these modes. One where the user can add folders and structure the work however they want and another where the program builds the structure based on the project content itself.

Functionality menus

Understanding a node based data flow is easy when the creator explains how it works. However, creating a data flow from scratch might be difficult. To lower the learning curve it would be valuable to introduce the nodes and how they differentiate from each other in an intuitive way. In Noodl users create a node by right clicking the workspace and select a node from a drop down menu. This feature also allows users to quickly type to filter nodes. This is a great feature to increase the efficiency for experienced users but creates a large threshold for new users.

We experienced that tools like Pixate and Node-RED menus are easier and faster to get a hold on. Even though we still had problems of what to place at which spot it was more intuitive and easier to learn by trial and error than watching a video tutorial.

Double coded functionality

As found in Node-RED and Pixate we believe icons combined with functionality names would be a valuable extension, especially to a tool with sensor integration. Sensor terms might be difficult to understand for most users and can be easily simplified with an icon.

Separated sensor functionality

We feel it necessary to differentiate sensors from other functionality. Sensor nodes tend to be more logically extensive and need more explanation of how they are used. A good idea would be to somehow give and explanation How the sensors are used in practice, rather than a technical description. This would also be a good way of inspiring users to new innovative ways of using the sensors.

Social nodes

When testing IFTTT we found that social functionality such as having a data flow sending a tweet or an email is a popular way of interacting. We find this functionality a great way of creating prototypes that seems more technically advanced, in a fast and simple way. The problem with this feature is that the tool will be dependent on other companies, meaning that if they, i.e. Twitter, Facebook, Gmail etc. change how their system works it would also demand hard-coded changes in the tool functionality. A counteract can be to somewhat limit the functionality to the most common social interaction. When using Gmail the prototyper could be limited to the functionality of sending a specific message to a specific email address.

Sensor simulation

Using sensor functionality should not require users to own the physical sensors. This might lower user creativity. A necessary feature is for users to be able to create the logical flow of the prototype whilst testing if it actually works through the tool. For a sharp trigger sensors this functionality could be an on/off button, and for a thermometer a draggable scroll to simulate temperature change.

Comments in flow

Quartz Composer uses a feature to comment single- or cluster of nodes. This is something that will simplify the understanding for users looking at the work of others. If the user in an easy way can explain their line of thoughts to others this will help all users that are working with the same, or similar prototypes. The feature could work by selecting an area containing nodes in the workspace. A text field is shown where the user writes an explanation about the functionality of the area.

Page visibility & orientation

From creating our exploration-prototype in chapter 5 we found that a big part of the prototype interaction is hiding and showing components. E.g. you have a screen with a button and when you press this button another screen will be shown. This is commonly done by either placing the thing you want to hide outside of the screen and then change its x-position value to make it visible or changing its opacity to 0 followed by 1 if it should be shown. The negative side effect is that that user might need to keep track of what components are placed at which position and set to which opacity.

We would recommend improved and more intuitive support for the user to program which components to become visible and in which way they become visible.

UI screen

A feature that is great for prototyping tools that are simple but very effective is the use of a real time updating view of the UI screen. This screen shows what the actual prototype look like and, by screen interaction, how the interactions feel. A problem with this is that sometimes users build prototypes with no visuals, for example with sensors, and then there are no clear visualization. Some other type of visualization has to be shown in those cases.

We feel this feature to be almost mandatory. It provides quicker design iterations, than only using an external device.

Different screen view

When the users are working and want to modify various components they might want to get a live visual update of the work. Instead of going through the main UI screen and locate the component or page you are working on it could instead use a separate screen where the component or page you are working on are displayed. This screen is sometimes redundant which could be solved by adding a toggle on/off button next to it to make it visible or invisible. However it should be displayed at the same time as the main UI screen since the component screen is only a visual representation of the component or page. If you want to try how the interactions work with all the other components you would have to go through the main UI screen.

Sharing

The ability to share your work to others and get feedback is immensely powerful since there is almost always someone who can explain your problems in a good way. Thus making the feature to share both your projects and parts of the projects with others would prove useful to all parts. To make this feature good a community has to be built around the tool. We have found that all the successful tools have a good community of users that helps each other when a problem occurs. For example the user want something to triggers when you get close to an iBeacon but do not know how to solve this. He posts this on the forum or another form of media asking for help. Someone responds by sharing his/hers solution to a similar problem which can be modified for the first project.

Another feature that is useful and used in many tools is the ability to create teams. A team is basically different users that cooperate and work on the same project at the same time. This will simplify bigger projects where you can divide the work amongst the team members.

6.1 Comparison to related work

The work by Fogli et al. [3] was the work we found to be most similar to our project. Their tool is solely focused on the sensor interaction which our tool is not. Their approach is more of a block based programming when dragging and dropping events which makes some of the functionalities we have listed here not applicable. However they have seen the value to interact social nodes with IoT as we have. To be able to test that the built scenario works they also have a test phase. It is however unclear if you can simulate the sensors in their program or not which we believe is something very important when working with sensors.

7 First design iteration - Tool paper prototype & usability test

This chapter present a tool paper prototype that has been designed based of the knowledge gathered from explorations in the previous chapters, especially the conclusions of chapter 5.

The paper prototype are usability tested on users from the three target groups stated in chapter 1 using the RITE method explained in chapter 2.

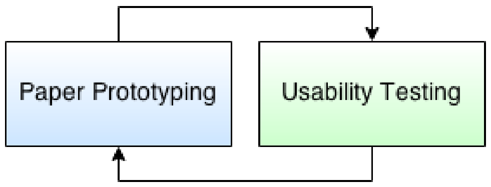


Figure 7.1: Overview of activities in the first prototyping iteration.

7.1 Paper prototyping

A paper prototype or a Lo-Fi prototype is a designed prototype with the use of paper that acts as the different menus and buttons. The advantage of this type of prototype is that it is easy to change while testing if some of the design has to be changed. It is also cheap to create since the prototype does not need to be coded to get the things exactly where you want it. Figure 7.2 shows how a paper prototype could look like [53].

For this iteration we chose to have a paper prototype that is a representation of the tool on paper. The main idea is to make it look like the tool and have the same functionality but when interacting with the design a test leader switches paper and actions depending on what is interacted with. This also helps the tester to be more honest about their opinions since they will feel it is not a finished as a Hi-Fi prototype.

7.1.1 Tool prototype walkthrough

Figure 7.2 shows how the paper prototype looks like when the first scenario (see below) is finished. The A3 paper sheet represents the screen on the user's computer. The things on the left of the sheet are settings for the different nodes that the user can modify. E.g. the one on the upper left represent what will be shown when clicking on the timer node. The green circle on the left is a physical representation of a force sensor with an id on its backside. The zone in the upper left of the sheet is the component folder menu, here all the created components will appear. On this zone there are three +-something buttons, these will create a new component and depending on which the user choose different nodes will be created in the new component. Under this zone there are the node menus. The first part exists of “normal” nodes and they are grouped into three categories that the user can switch between. The menu at the bottom left is there for the sensor nodes. These are categorized into two submenus, which consists of different nodes depending on the category.

To create nodes the user will press on the piece of paper that has the name of the node the user want to create. When it is pressed a new piece of paper representing the node will be available on the sheet. To connect the nodes (seen as the yellow arrows on figure 7.2) the user says that they want to connect these two nodes from off to start in the case of from force sensor to timer. They could also do this by dragging their finger from one of the nodes to the other.

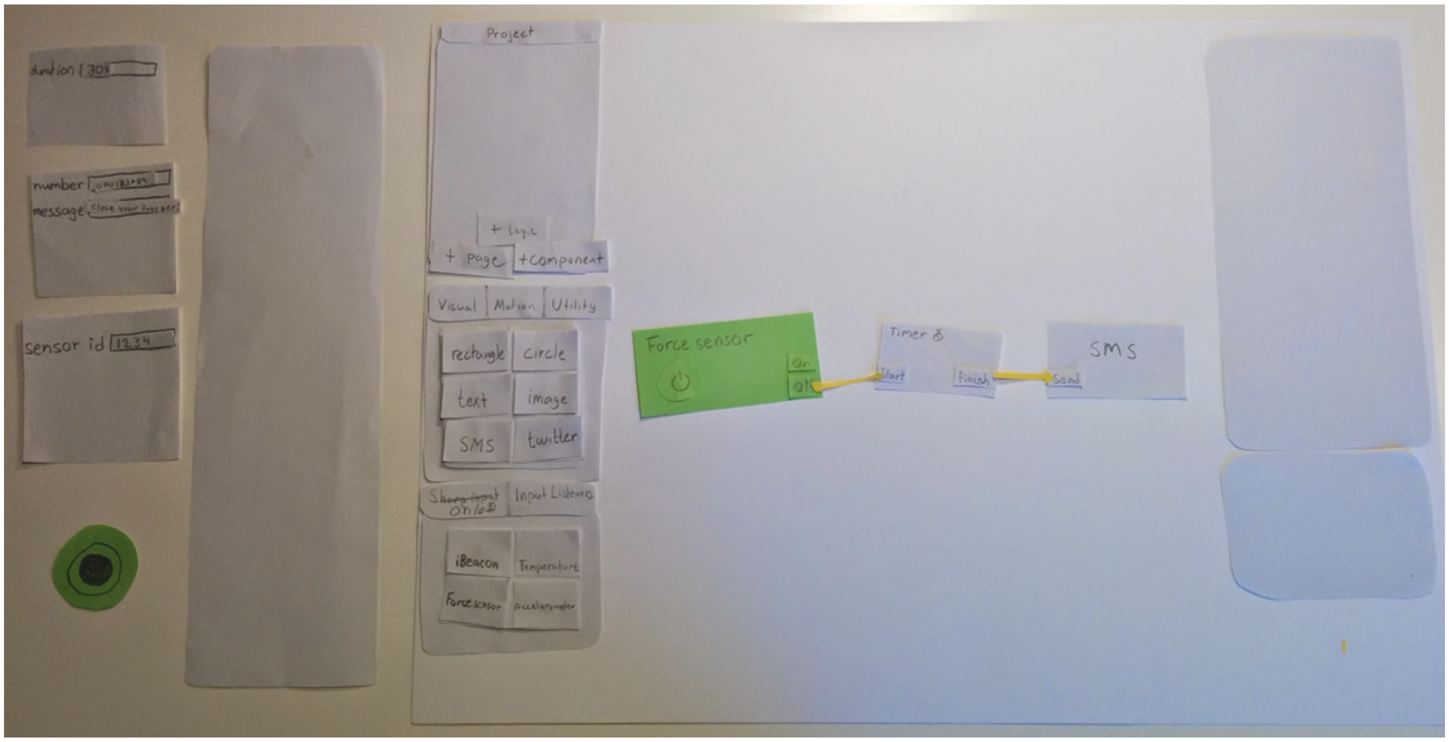


Figure 7.2: An overview picture of the paper prototype when the first scenario is completed.

On the right side of the screen is the screen representation of what visually will appear with this prototype. This screen is linked to the main component (or the one that are marked to be shown). Since no visual nodes have been created nothing will be shown in this scenario, in the second scenario however there will be things showing on this screen. Under this screen is a component screen, the things that are shown on this screen is the current component the user are working with. Since both examples only contain one component this will show the same as the other screen.

7.2 Usability testing

To test these functions a Lo-Fi prototype were made and tested on real users.

7.2.1 Method

For this usability test we used rapid iterative testing. The reason for choosing this method is from a recommendation by Jen Ignacz, UX Researcher at Topp. The method is suited for the early paper prototypes we did during this design iteration. If we encounter similar problems the users find during the tests we change them in between the tests to observe the difference. The test persons were asked to think out loud about how they interact with the tool. The tests were filmed with both audio and video to easier analyze the results from the tests.

7.2.2 Setup

The test was divided in three rounds of target groups:

The first round we tested on three Experiences Programmers. These users are all last year students of computer engineering with deep understanding of programming terms and logic.

The second round we tested on four Limited Programmers. These users are aware of basic programming logic. All of them are first year engineers, but from different programs.

The third round we tested on seven Non Programmers. These users were a mix of students of different education. We tested users from; Economical science, Chemical Engineer, Water engineer, Social science, Industrial designer and communication. None of these users had prior knowledge or interest in programming.

In the test we switched between two similar but slightly different scenarios. No user did both scenarios. All of the tests had a camera on the side of the user, which filmed the paper prototype as well as the hands of the test person.

Scenario 1
Your task is to create a prototype of a freezer door alerter. If someone opens the freezer door, a physical Force Sensor is triggered. If the door stays opened for longer than 30 seconds you will *receive a text message telling you to close the door.*

Scenario 2
Your task is to create a prototype of a freezer door alerter. If someone opens the freezer door, a physical Force Sensor is triggered. If the door stays opened for longer than 30 seconds *a circle on your screen will switch color from green to red.*

The users are testing a paper prototype of the prototyping tool that are almost completely interactable, meaning that the testers are able to interact with functionality and enter menu not directly of importance to the scenario they are to test.

7.2.3 Purpose

The purpose of this usability test is to try the method of visual programming with nodes and the way they are connected. This will also tell how well users understand the logical flow of nodes without any real introduction. We also want to see the response from interacting with the node and sensor menus, which might lead to new concepts of sensors and node representations that might improve the design.

The purpose of the first scenario is to focus on the programming flow of "if this then that" logic. We mainly want to observe how users react when not using UI screen interaction. With the second scenario we want to combine the logic behind sensor integration and the structure of how to build an UI screen.

7.2.4 Results

Based on the test we found a clear distinction between how the three target groups we tested navigated through the tool. *Experienced Programmers* and *Limited Programmers* both had no problem of breaking down the scenario into knowing which objects they would like to use. They both generally clicked around the different menu options to find what which components they wanted to use. *Non Programmers* had slightly more difficulty of finding what they were looking for. Five *Non Programmers* said that they did not know what they wanted to do and were confused by many of the options in the menu. By trial and error they manage to find the objects they needed.

When it came to connecting the nodes, *Experienced Programmers* had a harder time understanding how the nodes are connected than *Limited Programmers*. All three of the *Experienced programmers* were looking for more advanced programming functionality to put on the nodes, which in our tool are run in the background and not needed for the user to specify. This was something that none of the *Limited Programmers* even mentioned. When the concept was explained, none of the *Experienced Programmers* and *Limited Programmers* had any problem understanding the concept of having node objects separated and drawing a connection in between.

We found a different pattern of structuring the logical data flow when testing *Non Programmers*. They all had a harder time breaking down the scenario in different parts. Instead they wanted the logical flow to be bunched together with the Force Sensor at the top, then the Timer in the middle and the SMS on the bottom. This pattern of logical presentation was something we saw when testing all *Non Programmers*, but with none of the other target groups.

In total, twelve users had a hard time understanding how the tool worked in the beginning. Once they tried by pressing different options and observing what it resulted in nine of the twelve users learned rather quickly. Meaning that they reused gathered knowledge of how the tool responds to perform the next step in their work process.

The first three that did the test felt like SMS and Twitter nodes were out of place in the visual tab of the node menu. After these three tests we moved the nodes to the utility tab that worked better, but still got remarks of not fitting in.

The button named +Logic was used by three users to see if this could show them nodes or a menu of some sort to do the logical things with. All of them were *Experienced Programmers* or *Limited Programmers*.

Nine had trouble understanding how the sensor menu and its different content worked. *Experienced Programmers* thought that sensor listener were functioning like listeners in java/android programming where they listen for an action to happen and then do something. There were also a lot of questions of what the different sensors are used for, especially iBeacon.

When asking *Experienced Programmers* if they could see themselves use this kind of tool for this type of scenario instead of coding all of them said they could. Even for larger more complex projects they thought it could be useful. *Limited Programmers* all said that they liked it and that they could see themselves using it. *Non Programmers* were more divided. Four said that it was quite easy and that they were impressed that they had managed to program the scenario. The others said that they probably would never use this tool, mainly due to lack of interest.

7.2.5 Discussion

A valuable insight gathered from the tests is the idea that *Non Programmers* differs in the way they want to logically structure the visual programming. Some said that they did not understand why they needed to connect a timer instead of telling the sensor itself to just wait 30 seconds before sending a SMS. When putting the nodes on top of each other they said it to work as a timeline that would be run stepwise. What they subconsciously did was to use Block Programming instead of Flow-based Programming. We think the reason behind this could be that they have no knowledge of Object-oriented programming.

Structuring the solution of the two scenarios in this way is not at all a bad thing to do. It might even be better and more intuitive. The scenario they tested both have a clear timeline over which object that triggers the next one. However, the complexity of this method of programming becomes harder the bigger the system gets. Aside from this fact there still might be a valuable point of keeping the method of Block Programming in mind, seeing that *Non Programmers* using this type of tool in many cases want to create smaller, possible timeline viable prototype solutions. This could be a good solution for lowering the learning curve.

7.3 Conclusions - Tool paper prototype & usability test

We believed that the node based method of building data flows, similar to what Noodl and Form uses are easy enough even for Non Programmer to create smaller data flows with. From usability testing the method we found that *Experienced Programmers* and *Limited Programmers* had no problems, but *Non Programmers* had a hard time. Instead, we saw a pattern in a different way, *Non Programmers* tend to imagine and structure their solution. We also found our way of structuring the tool menus to be unclear.

8 Second design iteration - Hi-Fi prototype & concept evaluation

In this chapter we design a new concept of building data flows based on the results and conclusions from chapter 7. The purpose of this concept is to make prototyping of sensor scenarios easier for Non Programmers. We also create a Hi-Fi prototype to test and evaluate whether the new concept is easier to understand.

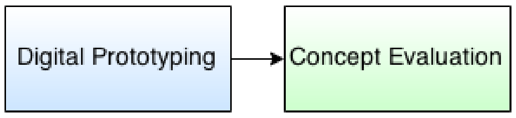


Figure 8.1: Overview of activities in the second prototyping iteration.

8.1 Digital prototyping

For this design we chose to use a Hi-Fi prototype. The term Hi-Fi prototype is used in many different ways and the definition vary from person to person. Most of the tools in chapter 4 like Invision and Noodl have their own view of what they consider Hi-Fi.

For our design we chose to build a Hi-Fi prototype with functionalities we wanted to test out in a realistic way. This means that the nodes are interactable and the user can move them around and connect them to other nodes with some restrictions. We did not build it pixel perfect since the purpose was to see if the concept worked well or not. This also means that the user cannot edit the nodes as they want, e.g. they cannot change the color of a circle. The prototype was mainly coded in JavaScript with the use of jQuery.

When developing the graphical layout for our prototyping tool we chose to not create a completely new layout and interface. This saved us a lot of time in design iteration two. Our prototyping tool was therefore greatly inspired by Noodl. This decision was based on that many features we had researched would fit in the Noodl environment. We also like the coloring and simplicity of their platform layout, which is a good place to start from. Note that the following figures are not Noodl, even though they look very similar.

Figure 8.2 shows a concept based on IFTTT which is a system that influenced to the design direction in our tool. The Non Programmers from the first test iteration wanted to build their system more timeline based than flow based. Their thought process was that first something happened then something else happened and so on. As seen in figure 8.2 the flow is more linear than it was in the first iteration and also more sequential. The feature that the nodes had slots where they could be added to was something that could help and support new users. The plus-button was something that was introduced which was an extension to the IFTTT that only has one condition. For more complex systems the multi condition becomes almost mandatory because there exists so many scenarios where different conditions leads to different results e.g. if you tap on a circle something happens and if you double tap another thing happens.

Figure 8.2 became the conceptual first stepping stone to the Hi-Fi design that were developed in the rest of this chapter.

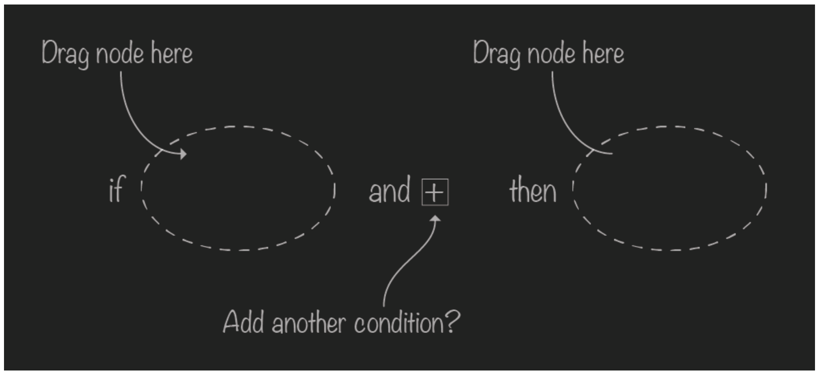


Figure 8.2: First conceptual design which inspired to the Hi-Fi solution.

As seen in figure 8.3 we have kept the interactable UI screen Noodl provides. The screen will present how the actual application screen and its animations and interactions look and feels. This provides faster design iterations. We have also chosen to expand this feature to include a second screen for the components the user is currently working in. This will help the user to faster test their flows without losing the overall view of the application. This screen is sometimes redundant for smaller projects, which was solved by giving the user an option of toggle this screen view on or off.

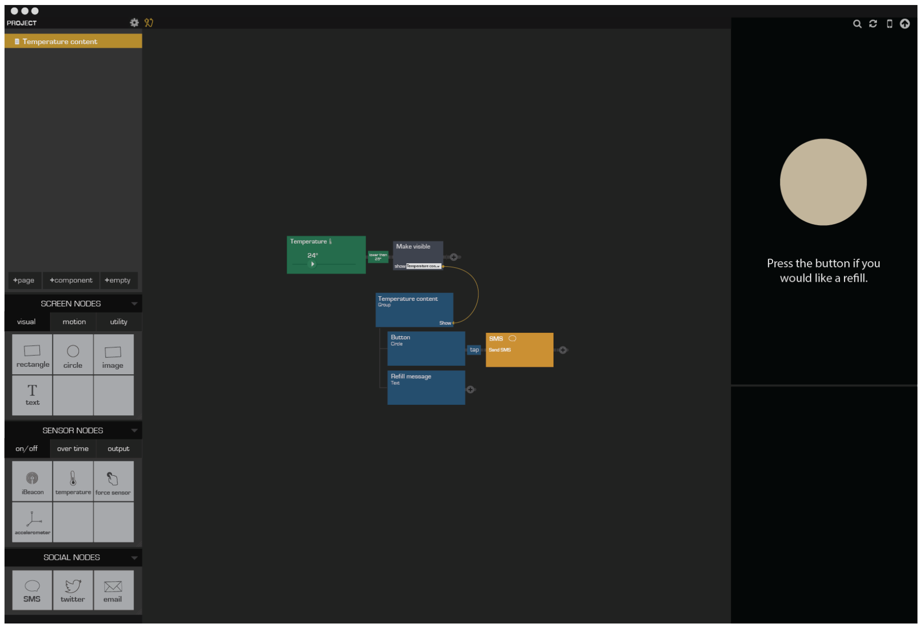


Figure 8.3: Overview picture of the digital prototype.

From the first design iteration we found that Non Programmers had a hard time understanding how to structure the data flow. All of them structured the objects on each other to create a perception of a timeline to know which order the objects are to be run. With this in mind, we developed a hybrid concept of both flow based and a more timeline based (block programming). This because a tool purely based on timelines are not that good, seeing that user scenarios are often, but not always time dependent. In figure 8.4 we present an example of a data flow with three different timelines and a connection in between.

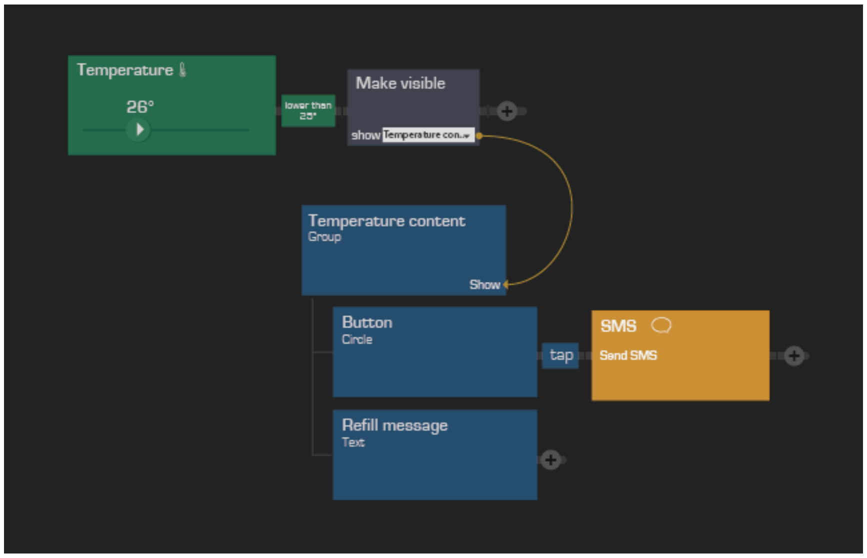


Figure 8.4: Digital prototype of the smart coffee scenario data flow.

When a user taps the timeline-plus, see figure 8.5, they will see a menu containing all possible next steps on that specific timeline. This will assist users to build quick data flows as well as support less experienced users understanding what to do next.

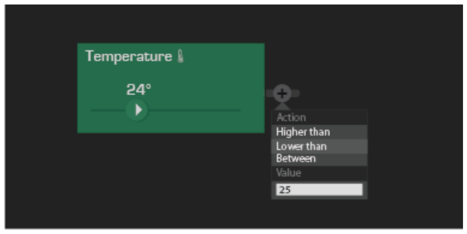


Figure 8.5: Timeline-plus functionality.

The smaller nodes on the timelines are interactions that wait and listen to their command being called. They work as Boolean values, telling the timeline when the data flow are to start. The play button on the Temperature node work as a manual trigger for a virtual sensor. Using this lets users create projects and test the flows before setting up hardware. The Make visible node seen in figure 8.4 is a new feature we developed for users to easily keep track of objects that are shown at which time. We also experienced a lot of users not understanding what opacity means from the first design iteration, which makes this feature even more important. When linking Make visible to group we give the user two options. A user can drag from the node menu to the target node, as done the first design iteration. On the Make visible node, the user is also able to choose which object to show from a list of possible choices, which will create a connection.

The menu system is similar to the one we created in paper prototypes but has been developed according to feedback from the first design iteration. We have divided nodes into three menus; *Screen Nodes*, *Sensor Nodes* and *Social Nodes*. The content of these menus can be hidden by pressing the triangle to the right of the title. A complete list of nodes needed for our tool is not decided. A list of this sort could change how the menu system is structured.

Screen Nodes menu contains nodes that is in some way connected to the screen. Its content is divided into three subcategories; *visual*, *motion* and *utility*. The nodes are all double coded with an icon and text for inexperienced users to faster understand what the nodes do. The blank nodes in the three following figures are just placeholders for the menu content to be expanded on.

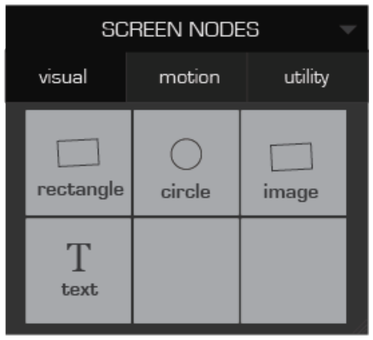


Figure 8.6: Screen Nodes menu.

Sensor Nodes menu contains nodes that in some way could be connected to hardware. Its content is divided into three subcategories; *on/off*, *over time* and *output*. *On/off* contains sharp trigger sensor nodes which are all manually triggered by pressing a button on or off. *Over time* contains input listener sensor nodes whose constant data flow could be interesting for a project. *Output* contains output sensor nodes. It is important to note that both *on/off* and *over time* can include sensors nodes using the same type of sensor hardware, but using it in different ways.

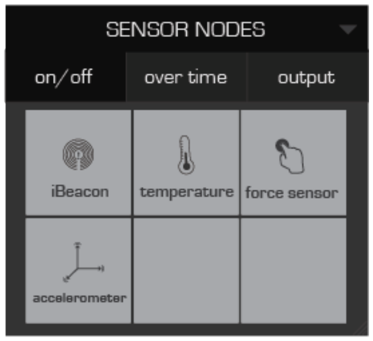


Figure 8.7: Sensor Nodes menu.

Social Nodes menu contains nodes that in some way lets the system communicate outside itself. These nodes and what the user can do with them are limited to basic functionality i.e. sending an email to a specific address.

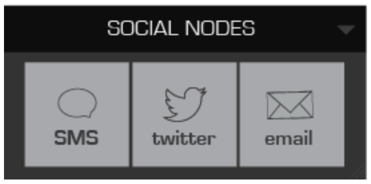
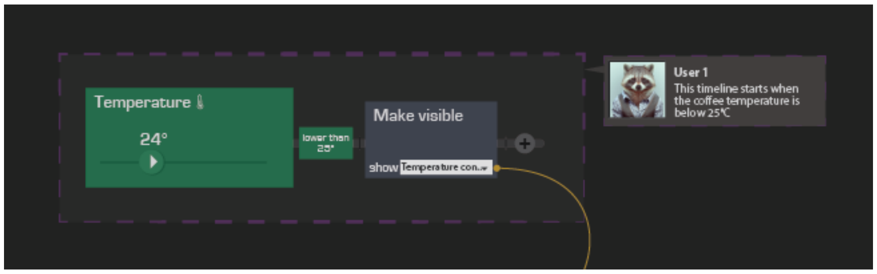


Figure 8.8: Social Nodes menu.

Figure 8.9 presents an example of how commenting could be neatly included in the flow. The user selects an area of the data flow to highlight and gets to write a comment explaining what that particular part does. Why the comment has a username and profile pictures is for creating a bigger community of users. Doing this might give attention to creative users to share more, as well as give support for inexperienced users. This will also make it easier for teams to cooperate and create prototypes with more than one person creating the whole flow.



User 1
This timeline starts when the coffee temperature is below 25°C

Figure 8.9: Commenting

For the user to get a better overview we chose to categorize components. Their basic usage is all the same, but what new window starts with is different. Creating a new *page* creates a new component with a group node and rectangle node with the size of the chosen screen. Creating a new *component* grates a new component with a group node. Creating a new *empty* creates a new component with no content. All categories have different icons shown in the content menu and will be automatically structured according to their dependencies.

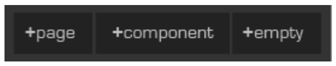


Figure 8.10: Buttons to create a new page/component/empty window.

8.2 Concept evaluation

We tested the conceptual changes on actual users. This was made by using a prototype walkthrough.

8.2.1 Method

For this concept evaluation we used prototype walkthrough (see Chapter 2.4). This method is suited for Lo-Fi, as well as Hi-Fi prototypes. We asked the test persons to think out loud and try to explain what they think about the tool and why they interact with it as they do. These tests were filmed and both audio and video were recorded.

8.2.2 Setup

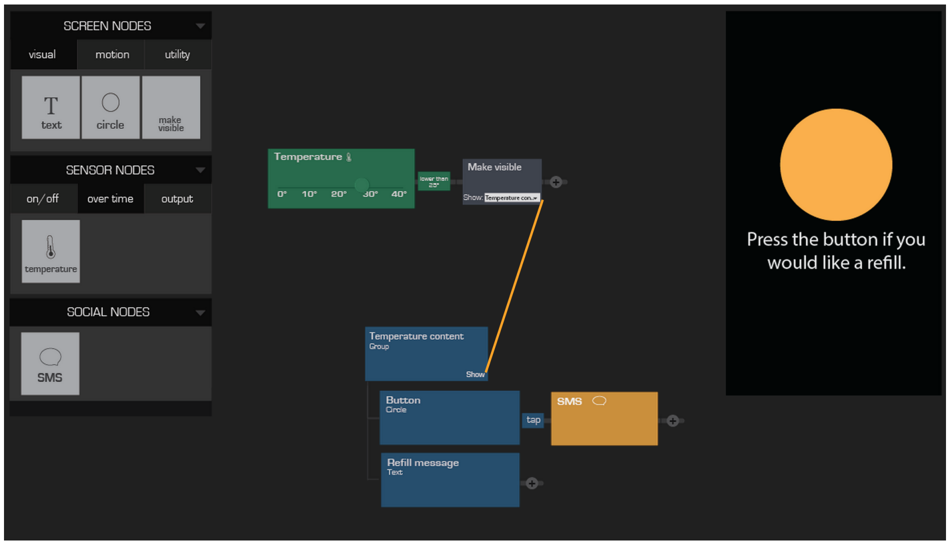
The test was divided into two rounds, each containing all three target groups. The first round we tested on users that have previously tested and given feedback on our Lo-Fi prototype. One limited programmers and three Non Programmers were tested. The second round we tested new users that have not seen nor heard anything about our tool before. We tested three Experienced Programmers, all last year computer engineer students. One Limited Programmers, first year computer engineer student. Two Non Programmers were tested, both Social Science students. A camera was placed diagonally behind the user and recorded the screen where the test took place. It also recorded what the person said.

In the test all users received the same scenario to build.

Scenario
You are to build a prototype of a coffee customer service for your local café. The bottom of every coffee mug contains a small temperature sensor that is connected wirelessly to this prototyping tool through the Internet. In the tool workspace you are to build the logical data flow. When the coffee temperature is lower than 25°C, the application will show a button and a text. When the

customer presses the button, a SMS will be sent to the barista telling him or her to give the customer a refill.

The users tested a Hi-Fi prototype of the prototyping tool that are almost completely interactable, see Figure 8.11, meaning that the testers are able to interact with functionality and click everything they should be able to in a real version of the tool. Many selections are however limited to the functionality of this specific scenario. An example of limitation is the menu that only contains the specific nodes needed for the scenario. In this prototype we also fixated temperature and group node to make it easier for us to code the nodes to snap on timelines. The Prototype is created in JavaScript and when testing it is working through a web browser.



Press the button if you would like a refill.

Figure 8.11: The Hi-Fi environment that were tested.

8.2.3 Purpose

The purpose of this concept evaluation is to try the method of combining the way nodes are connected by using timeline as well as linking arrows between different timelines and nodes. We want to explore how the timeline design are perceived and used by the users.

The scenario is similar to the previous test but is slightly more advanced. The reasoning behind this was to pick a scenario that could be built using two different timelines rather than just using one.

Instead of testing the same amount of users from every target group we chose to focus on experienced programmers and Non programmer. This is based on us thinking the changes we made to the concept will change the feedback outcome from these target groups more than limited programmers. We also chose to not only test new users, but to also test users that have also tested and given feedback on our Lo-Fi prototype. This is mainly to see if they think the new way is better, worse or equally good.

8.2.4 Hypothesis

Before conducting our tests we discussed what we thought would happen. We believe that users will not understand the concept of creating flows from the timeline-plus at an instant. We expected them to first analyze the menu by pressing all the buttons. When dragging nodes around they were to see that snap areas become visible and drag the nodes to their places. Finally when they could not do more node interaction they will start looking at the timeline-plus buttons. The users might be able to easily connect Make Visible to the group node but only a few of the test persons will understand why.

If our core idea is proven not to work better than our previous test we will have to do a third design iteration with a new concept or have the flawed functionality changed.

8.2.5 Results

Based on the test we could not find a clear difference between how the three target groups navigated through the tool. Compared to our test in Iteration 1, the problems users had was not as affected by their programming experience. Nor was there any distinct difference whether they had tested the Lo-Fi prototype before. Most of the following results will therefore not be written from a particular target group's point of view, if not stated.

Nine out of ten completed the task successfully with little help or no help at all. Eight users liked how the timelines worked and they felt it intuitive to add nodes to them, either using the plus or adding nodes from the menu. Even though some did not know what the timelines were used for, they added objects to them to observe what happens. Three found it hard to understand that the timelines originated from a specific node. Nine found the node placing of horizontal timelines logical, but one Experienced Programmer felt slightly restricted in not having the option put the nodes wherever they wanted. Two out of five Non Programmers found the restriction to be a good thing since it helped them to only focus on the task and not how to structure everything. The others three did not comment on the restriction.

Eight users started by creating a Temperature node. The user that instead started by creating the circle interaction, said when having completed the scenario that he/she rather should have started with the Temperature node.

Nine users had a hard time knowing when they were finished building the scenario. Seven found it reasonable when we told them they completed the scenario, but two were surprised.

The plus function was something that varied a lot from user to user. Five users found them useful and only used them when adding nodes and functionality, whilst the other five saw no use for them or did not even know they were buttons. The users that did not know if plus function was a button got confused by them and had to get help to use them. Two users wanted the plus button to disappear when they had interacted the make visible to show the group node. This was mainly because they thought that "this part is done and now I start on a new" and did not want to add more functionality to that timeline.

One user that used the node menu instead of plus felt it necessary to receive more clear feedback when something are put in a correct position. When adding a Make visible to Temperature timeline it was unclear if the positioning on the node really would work.

Eight users had problems with understanding the group node and its functionality even though the test leader explained that it were linked to the screen people still found it hard to use. When asked about it after the test, seven users either did not understand the group at all or found it redundant. They wanted a more clear explanation of what the group nodes function actually was and how it is used. The representation of grouping under the group node (see figure 8.4) was commented by a few. The users that commented on them said that it was good but maybe there could be a better contrast between them and the background to see them more clearly.

All users misunderstood the temperature scrollbar that should represent a manually regulator to what temperature the thermometer will show. This was to test the system without waiting for the thermometer sensor to get to a specific temperature. All users that integrated with the scrollbar thought it was to set which temperature it should be lower or higher than.

The make visible node was something that eight users found hard to understand and what function it actually had. Since it only had one function, which was to show the group, eight users just clicked on it and clicked again and they got the correct result without even reading the text. After the connection had been made they got surprised about what happened and did not understand what they had done when there appeared a line from the make visible nodes show text to the groups show text. Three users thought this connection was unclear and they did not know what had happened or if they were in fact connected or not. Two found the connection to be was clear and liked the way it was represented.

Our new concept of combining block-based and flow-based programming will put some constraints on the dataflow. Two of the users said they wanted more freedom when deciding how to build their prototype than our tool gave. However, our demo was limited in a way that the users could not move the temperature and group node that might affect this impression. Four of the Non Programmers said they liked the restriction because it helped them to find the solution easier.

When being asked how they would explain the tool for a friend or a colleague, we found a slight difference in answers between the target groups. Three out of four Experienced Programmers and Limited Programmers explained it as a "prototyping tool for visual programming and sensor integration". Four out of six Non Programmers described the tool as "an easy way to program using pictures".

8.2.6 Discussion

Our main goal with the test was to see if the new concept of timelines would be an intuitive solution for every target group to provide an easier way of solving tasks with the tool. Compared to the tests in iteration one, we found a much higher success rate in iteration two. This time we almost never had to correct or help for the user to complete the task. A thing to keep in mind is that the tool environment for this test is more restricted thus pushing the users to focus on the actual flow of nodes. This could be one reason for better success rate. Based on not finding any clear differences in success rate between the target groups we find the concept of timeline-flows much better for *Non Programmers*. We also see that the new concept is as good, or possibly better for *Limited Programmers* and *Experienced Programmers*. This was harder to analyze because these target groups managed to complete the scenario about equally fast in iteration two as they did in iteration one. From asking test users that tested both iterations, all but one said that they favored timelines regardless of programming experience.

Why many users had a hard time knowing if they completed the scenario or not may be affected by them not creating a prototype of their own idea, but instead a short text description. A prototype is almost never finished and could always be changed or build further. It is therefore hard for them to understand to what extent the prototype should be. When being told the scenario was completed almost everyone said they understood how their flow could work and also be able to explain to a friend how their flow works.

Even though about half of the users did not understand the timeline-plus interaction we found the results positive, based on our hypothesis. When you are new to a system it is always easier to press what stands out, being the node menu in our setup. When seeing all needed nodes in their workspace they saw no need for creating anything else but to instead try moving the nodes around. We still believe this functionality to be an effective complement to the node menu but could require a short tutorial for the users to fully understand. We can also improve the button feedback for the timeline-plus to feel more as a button as well as find a way to distinguish timeline-plus for timeline-triggers and timeline-plus for adding new nodes. It would also be a good feature to be able to close timelines, thus hiding the possibility to add more nodes. This would make the flow look more finished. We will also need to improve the feedback of a node being put on a correct spot on the timeline.

We knew that users would have a hard time understanding why they need a group node to put visual objects on the screen, so we started each test by explaining what it represents. When starting to build they still seemed to not fully understand why it was needed. However, scenarios using screens is a bit more advanced and could require a tutorial on how it functions. Seeing that all users misunderstood the functionality to manually set the temperature sensor tells us that a change had to be done. We will have to find a way of hiding the functionality until the user are sure they want to use it. Make Visible is a function suited for bigger systems and perhaps redundant in a smaller size scenario like the one we tested. *Experienced Programmers* and *Limited Programmers* found this node more useful than *Non Programmers.* They felt no need for it, wanting to put the group directly on temperature timeline. We will look into how to make it easier to create small scenarios.

Experienced Programmers and Limited Programmers saw the system as visual programming with sensor integration. The Non Programmers saw it more as an easy way to program. Since the users that know programming is more aware of what happens and how the functionality in the nodes work they value the visual programming and the sensor interaction more. Compared to the Non Programmers that just see it as an easy way to program things.

When building this type of scenario, our tool concept is easy to use for all the target groups. However as the complexity increases, more is required from the users. The logical connections between interactions and actions is where the users have to think of solutions how to tackle their problems and how to solve these. In these scenarios we believe that a more experienced programmer will excel since they are normally more used to know what they need to do and how to solve problems. This is also where a good community comes very handy. If you have a problem as a *Non Programmer* you can try to find solutions made by someone who is more experienced and by that work around the increased complexity in the prototype you want to create.

8.3 Conclusion - Hi-Fi prototype & concept evaluation

We designed a new concept method of the way data flows are built to facilitate Non Programmers. The method keeps some elements from the first method but adds a way of thinking in timelines. We also added plus-buttons on nodes and timelines to help users know which nodes that fits which position. From concept evaluation we found it hard to tell whether the new method are more effective when creating data flows for *Experienced Programmers*, but found it way more intuitive for *Non Programmers*.

9 Third design iteration - Simplify sensor names & refining concept design

From chapter 7 and 8 we found that users had problem understanding the menu categorization of sensor nodes. In this chapter we discuss how to name and categorize sensors in a more intuitive way. There will also be some visual and functional changes to the tool concept that will be evaluated with a heuristic evaluation.

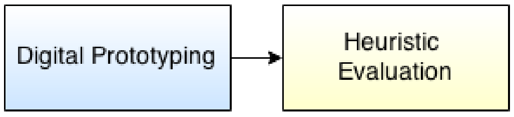


Figure 9.1: Overview of activities in the third prototyping iteration

9.1 Sensor nodes & menu categorization

In both design iteration one and two we found that node representation in the menus are unclear. The users did not know what differentiated the sensors types and were uncertain of where to look or what to pick. When asked what a particular sensor type does the user most of the times neither knew nor could make a guess.

This made us realize that it might be unimportant and possibly even illogical to represent the sensor nodes by sensor types. It might instead be more intuitive for the users to represent the sensor nodes by their functionality. We believe that most users have an idea of what experience they want to prototype, rather than owning a particular sensor type they want to build something around.

Figure 9.2, 9.3 and 9.4 presents how the nodes are now named and categorized in the tool menus. The nodes in these menus are commonly used functionality that might be of use for an inexperienced user. The remaining more advanced or uncommonly used nodes, as well as the nodes represented in the menu will also be accessible from a quick search menu when a user right clicks the workspace.

Figure 9.2 shows *Communication Nodes* menu with three submenus; *Sensor inputs*, *Sensor outputs* and *Social.* This menu contains nodes representing communication between the tool and something external.

The sensor nodes are divided into *Sensor inputs* and *Sensor outputs* instead of the three categories on/off, over time and outputs used in previous iterations. These changes were made due to users not grasping the difference of different input sensors. *Social* nodes were added as a submenu to *Communication Nodes* as they are representing functionality that will communicate from or to the tool via external servers.

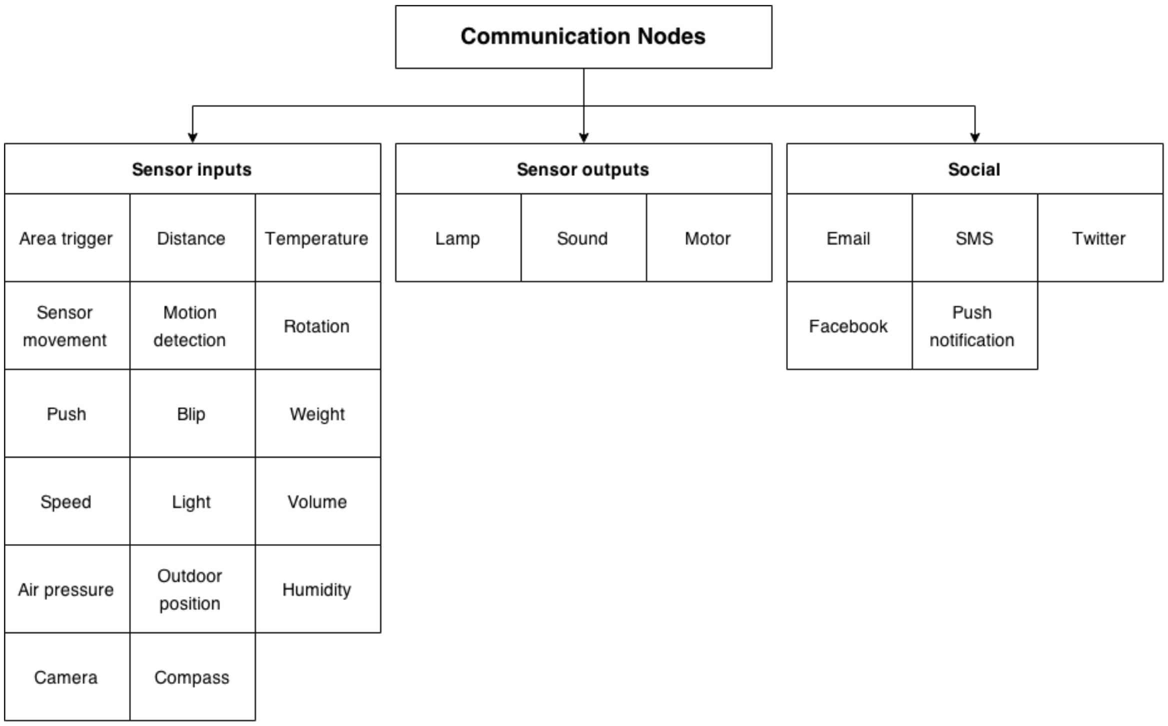


Figure 9.2: Menu structure of communication related nodes.

The node content of *Screen Nodes* menu are similar to previous iterations. A difference is that there is no submenu, but instead a menu of its own. The nodes in this menu are commonly used for creation of UI and should not be misinterpreted with other nodes.

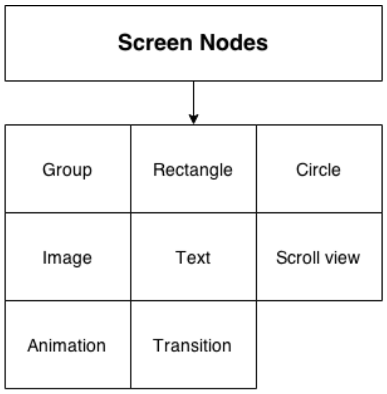


Figure 9.3: Menu structure of screen related nodes.

The node content of *Utility Nodes* menu are commonly used nodes that does not belong in the other menus. These are all functions which will be difficult for a *Non Programmer* to understand but might be important to learn and experimented with.

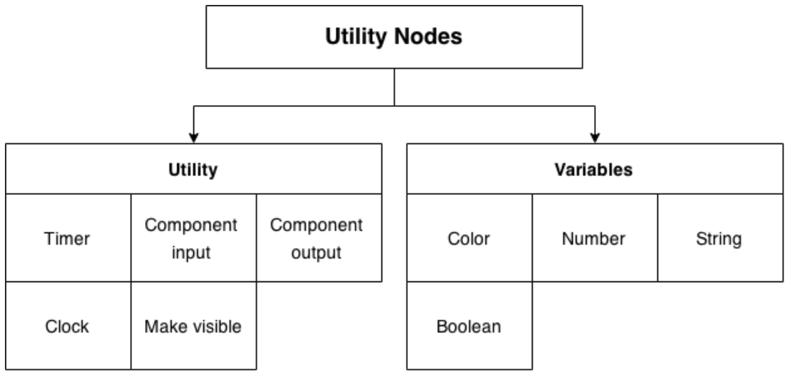


Figure 9.4: Menu structure of utility related nodes.

9.2 Visual and functional redesign

From the second design iteration we gained some valuable information of unclear visualization and functionality that we here decide to change in order to make it easier for the users.

There are now two different types of plus-buttons for adding nodes directly on the timeline. One is inserted within its specific node and represents the creation of a new timeline-trigger node, and thus a new timeline. These buttons are colored with the same color as their respective node has. The other plus-button lies on a timeline to easily insert nodes that suits that particular timeline. These buttons have the same color as the timeline have. The buttons have also been given more affordance of a physical button, as seen in figure 9.5.

Some of the users had trouble understanding the connection between *Make visible* and *Group* node. They simply felt no need for it in a scenario like this. The users thought it would make more sense to put the group node directly on the temperature timeline. We changed so that users can place the group directly on timeline, and through a drop down on the node be able to choose what to do with it. We still believe the *Make visible* node to be powerful but might be a more advance node to use when you are more experienced with the environment.

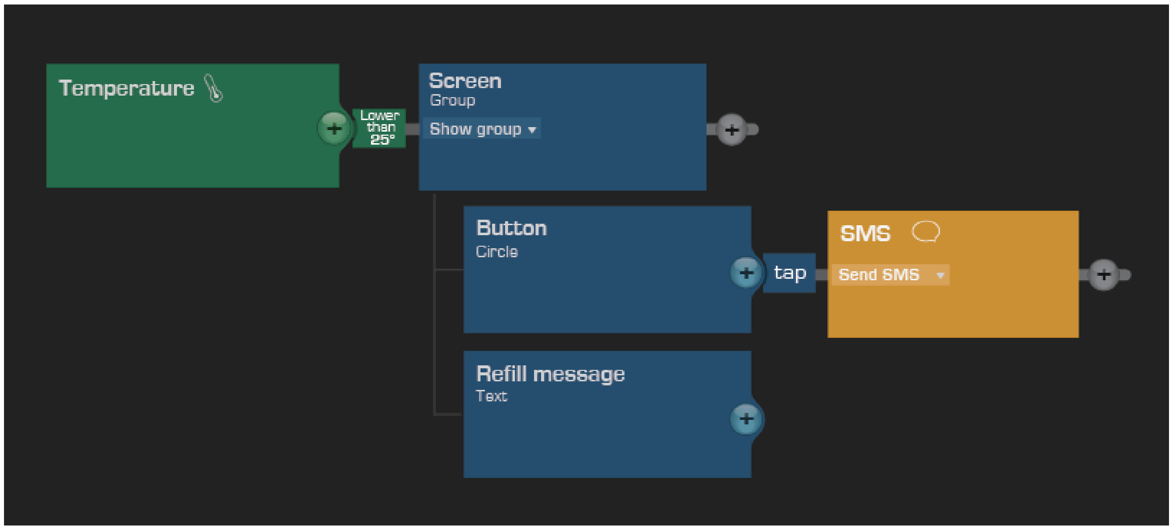


Figure 9.5: Visual and functional redesign of the node flow from iteration 2.

During the tests some users said they felt insecure of their dataflow actually being the solution they were looking for. They said it still had many loose ends they did not like. When comparing figure 9.5 and figure 9.6 you find that we added the ability to close timelines, making them look and feel more completed. Users can simply toggle this closure on and off by pressing the very end off a timeline.

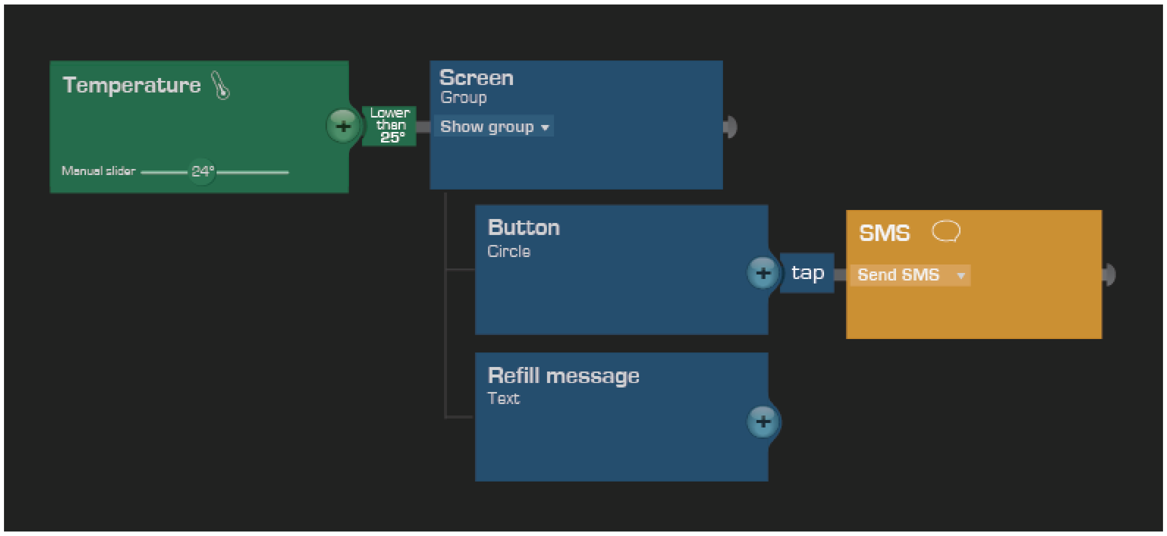
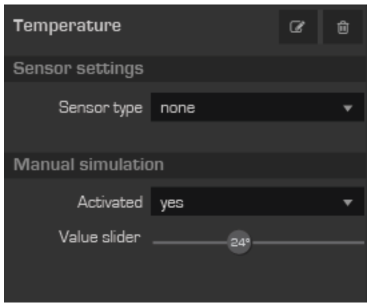


Figure 9.6: Visual and functional redesign of the node flow from iteration 2 when sensor simulation is activated and timelines demerged.

Every user that tested the prototype in in the previous iteration misunderstood the manual temperature slider. Instead of always being shown, the temperature slide bar are now hidden until activated in the node settings menu. This can be reached when clicking on a node, see figure 9.7. If this is set to "yes", a sliding bar will be shown if the node represents *Temperature*, as seen in figure 9.6.

This is applied to all the sensor nodes with different simulation depending on which sensor type. E.g. an *Area trigger* node will have fields where users type in devices that are in the area and the *Push* node will only consist of an on/off button. The menu bar in figure 9.7 will also change depending on which node is being modified in the same way as the actual node simulation in the workspace.


Temperature

Sensor settings

Sensor type: none

Manual simulation

Activated: yes

Value slider: 24°

Figure 9.7: Menu settings for the Temperature node.

9.3 Heuristic evaluation

For a usability evaluation of the third iteration changes, we chose the method of heuristic evaluation. Due to time limitations, we tried to find a method that required little or no external input and found heuristic evaluation to be a suitable method to base it on.

We made the evaluation separately by looking at the heuristics in appendix A and then compared our findings. Since the heuristic evaluation is based on only finding errors some small alterations were made to the evaluation. We took all the heuristics from appendix A and listed both how they were solved in our usability design as well as suggestions how to solve them if they were not solved. Even though it is recommended in [7] to use at least three persons in a heuristic evaluation we felt that two would be sufficiently to find any major design flaws in our tool.

Visibility of system status
When users interact with menu buttons and content in their workspace there will always be an action response. The interactable screen displaying graphics, animations and other interactions also keep users informed of what is happening in the system.

A thing the tool lacks is for users to know how sensor outputs responds to their build flow. This is something we need to explore further.

Match between system and the real world
The naming and structure of sensor nodes have been changed to be more familiar to the users. The concept of using timelines to represent data flow gives a more intuitive sense of how their systems work and where to place new nodes and connections.

User control and freedom
All nodes in the workspace are completely movable. Once a user move a node which has a timeline, the timeline and every containing node will follow the movement. If a user move a node contained on a timeline it will be removed from its position. The tool should also support undo and redo functionality, as well as an easy way of deleting unwanted nodes or connections.

Consistency and standards
The tool restricts functionality with similar names to be accessed where it is used. E.g. *tap* functionality for screen visuals can only be accessed from the specific node plus-button. This way the tool reduces the risk of confusing this functionality with the sensor node *Push*.

Error prevention
The UI and component view screen help preventing logical errors when working

with screen interaction and design. By adding a Circle they will get instant feedback that graphics of a circle has been added.

The tool currently lacks support for preventing logical errors when not working with the screen. If the user are to build an experience using only sensors they will not get any feedback of how their output nodes are responding to their flow. This is something we need to explore further. There is neither a limitation to which nodes that can be connected to which. This will also be reviewed.

Recognition rather than recall
Seeing that this is a complex tool that is able to build almost anything in UI design it will be hard for users to instantly know how the tool completely works. In a complete version of this tool it must include in-tool tutorials.

Flexibility and efficiency of use
The most commonly used nodes in this tool are presented in the menu. All of these, as well as more advanced nodes will be accessed by using the quick node search function.

Aesthetic and minimalist design
The used nodes contains only needed attributes and information.

Help users recognize, diagnose, and recover from errors
The tool currently has no displayable error.

Help and documentation
The tool currently has no specified documentation and will be required for a complete version of this tool. Video or in-tool tutorials will also be helpful.

9.4 Conclusion - Simplify sensor nodes & refining concept design

Instead of naming sensor node by their technical term we found an advantage in naming the nodes according to their functionality. This might help less technical users to understand which node to use for their solutions. From heuristic evaluation we also found a lack of visual feedback of when the data flow works as intended while using sensors.

10 Fourth design iteration - Sensor visualization & complexity levels

When testing our concept in the previous three iterations we chose to limit complexity and scale of the scenarios to prototype. For the concept to fit a testing environment it is important for the participants to not get discouraged nor for the test to take too long time.

This chapter challenges our concept with higher complexity levels of prototyping. What happens if a user wants to prototype something more advanced than pushing a button to send a SMS? We will also present a concept giving better visual representation of how the sensors used in the data flow works

The concepts presented in this design iteration is proof of concepts and will not be tested or evaluated due to time limitations. What we present are therefore not to be seen as an optimal solution, but as an example of how higher level of complexity could be solved.

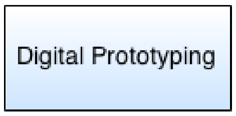


Figure 10.1: Overview of activities in the fourth prototyping iteration.

10.1 Complexity increase - Conditions

The scenario

To push the level of complexity we selected a scenario that puts more demands on the concept of timelines. In figure 10.2 a Smart Home solution is presented with responding Lamps with different colored light depending on who is inside a room. In this prototype one will be able to easily modify the colors of the lights for every family member in every room. It should also be possible to quickly extend and add more rooms with Lamps to the prototype. Each room contains a Smart Lamp and an iBeacon that keeps track of which users' devices are inside of its area.

E.g. the experience will function such as when the father, Liam, is in the living room the lights will glow blue. If instead the son, Martin, is inside the living room the lights will glow red. If one is inside a room during nighttime all Lamp dimmers will be set to 60% capacity. If the last person leaves a room the Lamps will first dimmer to 50%, and after 20 seconds be completely turned off.

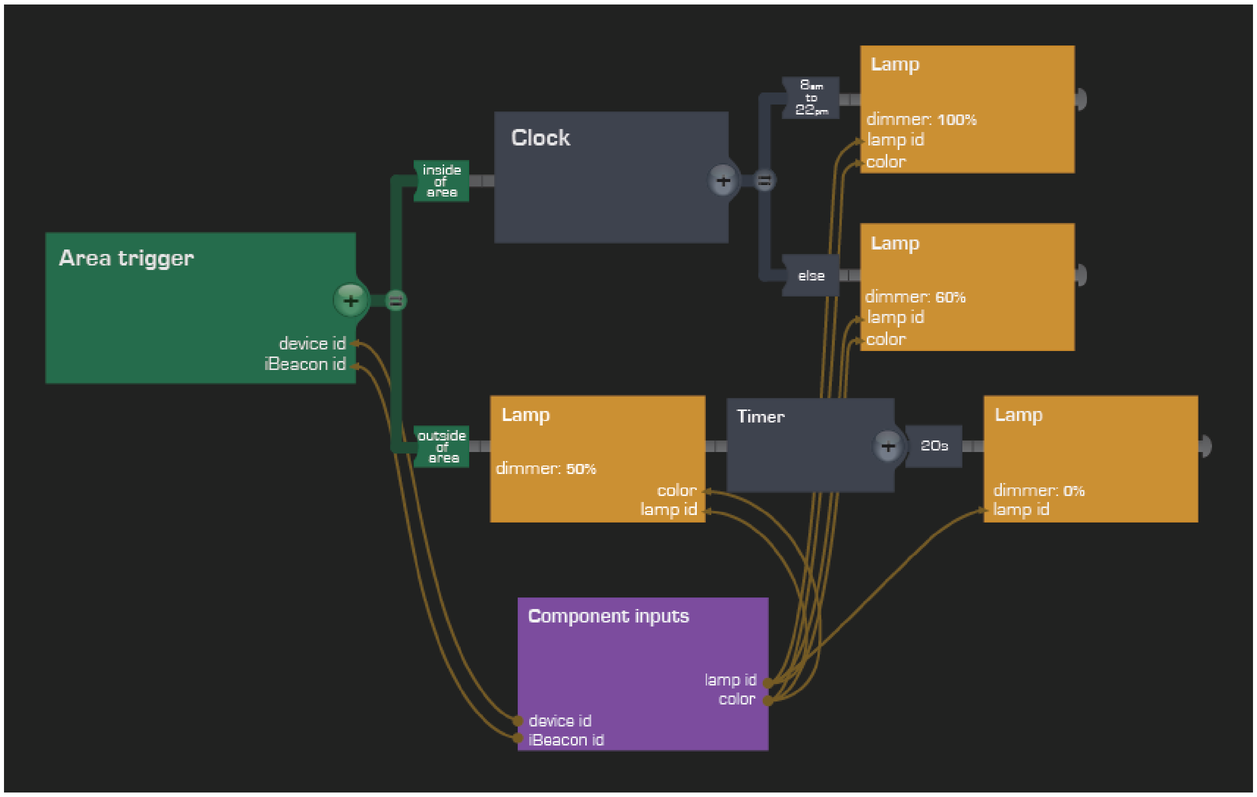


Figure 10.2: A Room component that sets the color and dimmer capacity of a lamp depending on the person inside the room.

Flow walkthrough

Figure 10.2 presents a room component in the system. The flow starts with an *Area trigger* node which has its sensor type set to iBeacon. The functionality of colored plus-button that is inside of the nodes is now able to add more timeline-triggers. This is necessary e.g. if you want different things to happen when you tap or double tap an image. Each of these will also create a new timeline. From this node are two different timelines. One that triggers if the room has one or more devices in area, and one that triggers when no one is in area. The inside area timeline contains a Clock node that keeps track of the current time. This in turn splits into two new timelines, one containing a Lamp representing its specific settings for dimmer capacity when it is daytime and the other for night. The second *Area trigger* timeline for no devices in area contains a *Lamp* with dimmer capacity of 50%, followed by a *Timer* node that waits 20 seconds. This is followed by another *Lamp* node that turns the lamp completely off. The small circle with an icon placed on the colored condition-part of the timeline shows the logical rules of the condition. E.g. in this flow only one of the timelines would work simultaneously.

The flow in figure 10.2 would work as described in the previous paragraph, but only for one room and one user. To be able to quickly create this flow for multiple different rooms and device settings we use a *Component Input* node. This node has output values specified in a higher abstraction of the system connected to node attributes. When this Room component is created in higher abstractions you enter which iBeacon id, device id, lamp id and color of the lamps in the flow. This is good for prototyping many things that flows with the same nodes but uses slightly different attributes.

It is important to note that *lamp id* is the same for every Lamp node in this flow. This means that every Lamp node is representing the same physical lamp, which might be difficult to understand by just glancing at the component. How this could be more clearly visualized will be presented in subsection 10.3.

We believe this level of complexity is something that would be hard for a *Non Programmer* to build from scratch without any help, but if you have an already build system and explain it for them we they would understand the flow and what will physically happen. With descriptive tutorials and good documentation we believe that this knowledge gap can be filled and let *Non Programmers* with an interest to learn, build these kinds of systems as well.

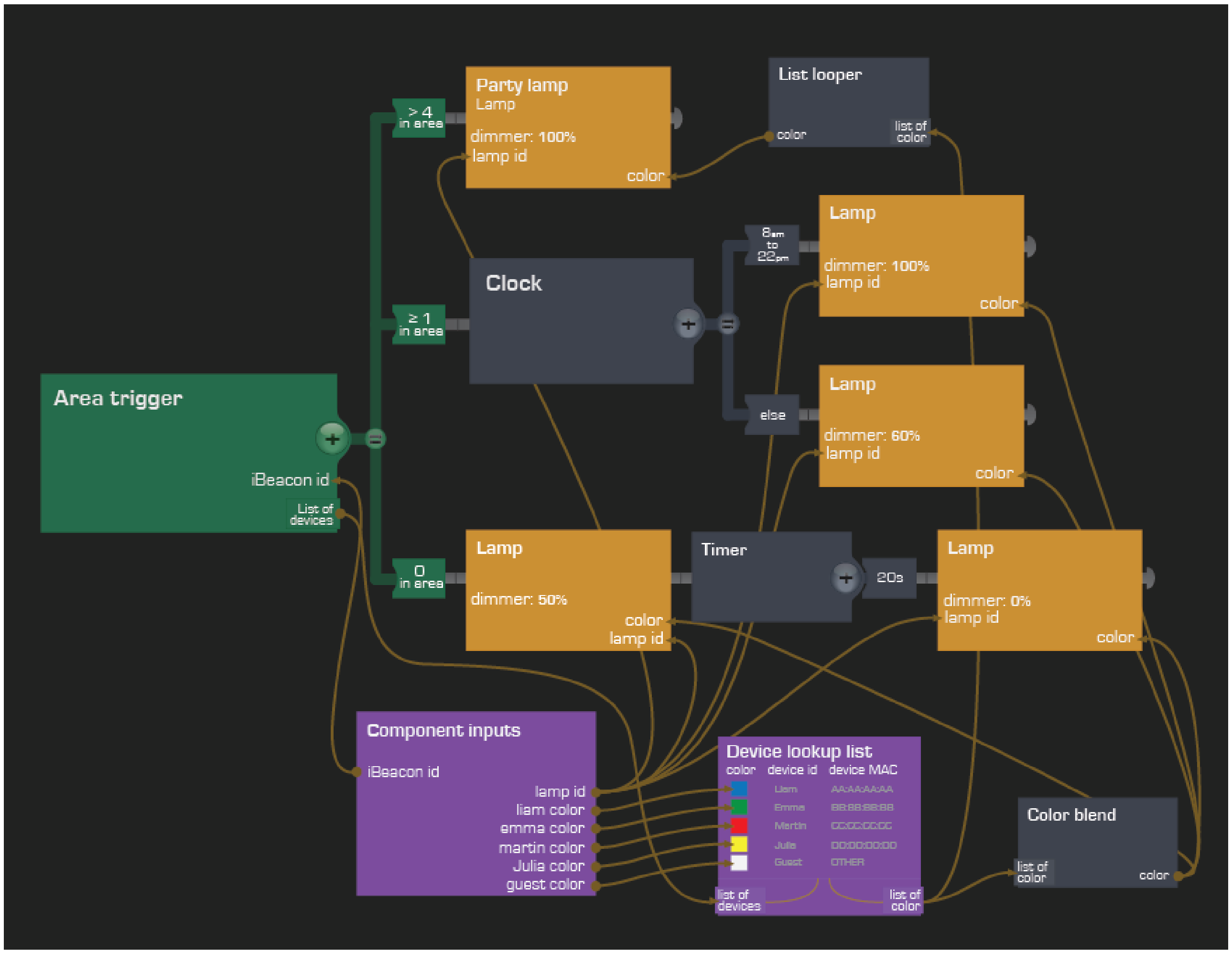
10.2 Complexity increase - Multi-user

To push the level of complexity even further we added multi-user functionality to the experience. When taking one user and one device into account, the previous scenario works, but what happens if two or more enters the same room? In the following solution we want the color of the room to be blended between the chosen colors of each user inside. If Liam with color blue and Martin with color red are in area the lights will be purple. We also want to add a Party Mode when more than four users are in area, whereby the lamp will repeatedly switch colors between every users chosen color.

The difficulty with this is knowing which user has which specific device and to simultaneously do similar, but slightly different things for different users. The solution in the previous subsection are solved by creating one Room Component for each user on each room. This solution requires a solution with only one Room Component for every room containing functionality for every user. So how will the nodes know which device relate to which user settings?

A solution is to introduce listing functionality. Sensor nodes, Area triggers for example will not only know how many devices that are in area, but also their MAC-addresses. We added a way to continuously send a list with these devices. By sending a list we also need a way to read and handle this data.

A *Device lookup list* is a node that allows attribute mapping. Users can sync devices and bind them to variables of choice. In figure 10.3 the user have already set which device belong to which person's name as a String value in a higher abstraction of components. The users also want to bind a color value to the device list, different for every room and user.



color	device id	device MAC
🟦	Liam	AA:AA:AA:AA
🟩	Emma	BB:BB:BB:BB
🟥	Martin	CC:CC:CC:CC
🟨	Julia	DD:DD:DD:DD
⬜	Guest	OTHER

Figure 10.3: A Room component that sets the color of a lamp depending how many and which person inside the room.

Figure 10.4 shows a data flow where Area trigger sends its list of current devices in area to the Device lookup list. The node checks matching color, and forwards a list of colors to a *Color blend* node which mixes the colors and send one variable of color to every *Lamp*. A *Device lookup list* can only receive and send lists but the list could also only include one variable. Every node cannot receive lists thus it naturally sets higher technical demand on the nodes. To get a value from a list a user first needs to connect it to some node that process the list and sends out the needed value, e.g. *List looper* and *Color blend* in figure 10.3.

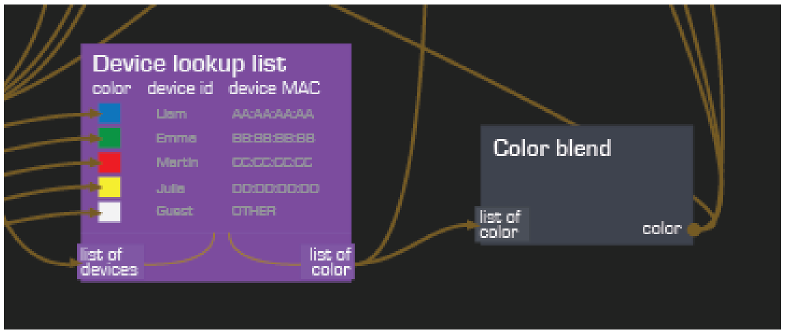


Figure 10.4: Device lookup list receiving and sending a list variable.

10.3 Sensor Visualization

When working with screen nodes such as a Circle or Rectangle these will appear on the UI screen and thus give visual feedback that the node works as intended. When working with sensors there is no natural graphical representations that will be shown on the screen and as result no practical feedback if they work. To give users better understanding of how the sensors work we developed a concept to enhance its functionality and relations to other sensors.

In chapter 6 we described the advantages of having a Component View as well as an UI Screen View. The Component View are now able to be changed to a Sensor visualization view. This view will show if the sensors are connected correctly. Figure 10.5 shows a visual representation of the sensor nodes used in figure 10.3 if they were not to be connected to anything. Since the *Lamp* nodes have no *Lamp id* they are assumed to be representing different lamps and presented with their respective dimmer capacity and a standard color.

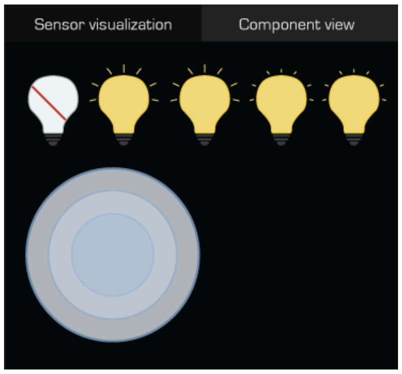


Figure 10.5: Sensor simulation of five *Lamp* nodes and an *Area trigger*.

Figure 10.6 presents how the sensor visualization will instead look like when the component is connected correctly as in figure 10.3. The *Lamp* nodes have now been assigned the same *Lamp id* and are connected to the area trigger. The *Area trigger* node has also been manually set to simulate that two users in area; Liam with color blue and Martin with color red. The resulting lamp color after a Color blend will result in purple light.

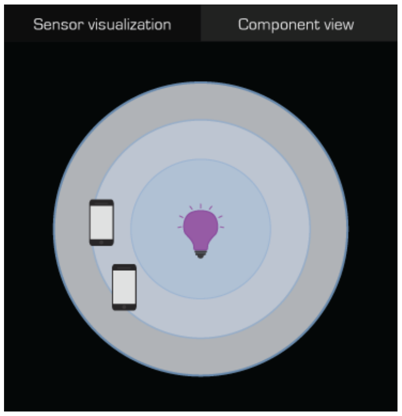


Figure 10.6: Sensor simulation of five different *Lamp* nodes with the same physical *Lamp id* and an *Area trigger*.

This visual simulation might help users understanding how the sensors are connected and how to interact with them. It also helps an external person that has not built the system to understand what will happen in the real system. For someone seeing figure 10.3 for the first time they might think there are five different lamps but when they see the visualization of the system they can see that there are only one lamp which increases the understanding of the flow.

10.4 Conclusion - Sensor visualization & complexity levels

We found the tool concept to be able to handle increased complexity levels of data flow solutions. However, by doing this there has to also be an increase of programming knowledge demanded on the user. A novice user will not be able to create the most complex systems, but there will always be a possibility to learn.

We also introduced visual sensor simulation to help users understand what happens when sensors are triggered and whether their solution works as they intended or not. This solution might increase the understanding of what the user have built and what errors they may have done.

11 Discussion

The first aim of this project were to explore existing ways of UX prototyping, different types of IoT scenarios and which sensors are of most importance when prototyping. We practiced a thorough Exploration phase where we focused on the first of our aims, to get a steady background of what is currently needed in the industry.

The second aim was to design new ways of prototyping UX when interacting with systems containing various sensors and multiple devices. In the beginning, we focused on creating a solution that will work for Limited and Experienced Programmers, as well as Non Programmers. This aim came to slightly change direction where we in the beginning of design iteration two switched main focus to Non Programmers.

11.1 General Discussion of the Results

The final prototype design of this project is a tool that lets Non Programmers build their own User Experiences in an easy way, but also lets Experienced Programmers build more complex data flows, all using multiple sensors and devices. The final prototype design is based on conclusions from the exploration and conclusions from each design iteration.

The design iterations

The following paragraphs in this subsection are a summary of what we discuss in the design iterations.

In the first iteration we had a hypothesis that the object oriented concept of building visual data flows used in Form and Noodl might be simple enough for Non Programmers understand for simple scenarios. We also wanted to see how users react to the menu system and their overall experience using our tool prototype. We found clear patterns that Non Programmers tend to see the logical solution to a scenario in a different way than Limited and Experienced Programmers. Non Programmers did not feel a need of being able to separate nodes and instructions, but instead wanted them to all be put together in the order of when they are used. We also found that the way we named and structured the menus were unclear to all users.

In design iteration two we designed a way of building data flows built on a combination of the concept from iteration one and to include timelines that might make it easier for Non Programmers. While testing this new method we experienced this concept to make the Non programmers more confident, while Limited and Experienced Programmers showed had no apparent difference.

In design iteration three we changed the way of approaching the node usage of sensor. Instead of naming sensor node by their technical term we found an advantage of instead naming the nodes according to their functionality. This might help less technical users to understand which node to use for their solutions. From both of the performed tests we found a lack of knowledge in sensor terms and what the sensors can do. This was something that all three target groups lacked. This naturally makes it hard for users to know which sensor node to pick. We believe this new sensor node structure will instead help users to know what to pick.

In the last design iteration we tested the new tool concept to be able to handle increased complexity levels of data flow solutions. By doing this, there has to also be an increase of programming knowledge demanded on the user. A novice user will not be able to create the most complex systems, but there will always be a possibility to learn.

We also introduced visual sensor simulation to help users understand what happens when sensors are triggered and whether their solution works as they intended or not. We believe this solution to increase the understanding of what the user has built and what errors they may have done.

Prototyping tool connectivity test

From the connectivity test we found that integrating a sensor into a system was complex and required quite some programming for it to work. This had to be done outside of the tool and then integrated into the tool by an external application that we had to write. This was just to get the phone to recognize if it was close to an iBeacon or not and only worked if the sensor were an Estimote. The problem we found with this is that if you want to implement support for a sensor you will have to implement it for the different brands of the sensors as well. E.g. an Estimote and a TI sensor have some shared sensors but the support for them will be implemented separately. This is one of the big downsides when building a tool for sensors. There is no real standard for sensors and they work in different ways.

The integration of sensors into our tool would require a thorough documentation about how to integrate the sensors into the tool since you can only come to a certain level when simplifying the integration. The users would most likely need to know some kind of id of the sensor and which type of sensor as well as which brand the sensor has. A good way to explain how this is done would be to add tutorials where it shows how to integrate a sensor into the tool and get it to work in a flow.

The concept model

The concept model we came up with in the second iteration is something that could be valuable in many ways. The hybrid of a flow based and a block based programming thinking with the mindset of a time line seamed to please many of the users that tried it. This is something we feel could be integrated in any type of prototyping tool regardless of the sensor integration. But since it has not been tested in a full-scale program and only in a Hi-Fi prototype with somewhat limited functionality we cannot really validate it. However with some tweaks and adjustments to the implementation of the concept we think it can be very promising.

Compared to Fogli et al [3] they have focused on a pure block programming version in their tool. This is because they have focused more on simple types of scenarios. In a sense when looking at simple scenarios our tool looks a lot like a pure block programming tool as well but with an increased complexity in the built system looks more like a flow based system. Since we have not tested their system it is hard to know which complexity level it can handle, but we believe that the last scenario in section 10.2 would not be possible in their tool.

The Hi-Fi prototype

The Hi-Fi prototype we built for testing the concept itself could have been more functional than it was. However this was because of the time limit of the project and going to the next level in Hi-Fi prototype would take about the same time as the rest of the implementation took. This is something that could be a problem when building a Hi-Fi prototype. That it takes longer than you think it would. Sometimes when you focus too much on the small details in the Hi-Fi prototype you can lose the focus of what the actual goal is for the prototype. E.g. if you want to show the prototype to a client to see if your design of what you have built is good, it would be good to make it as close as the final product as you can. Compared to when you want to test a concept as in our case the Hi-Fi must not be pixel perfect in its design, since you probably have to remake it anyway.

11.2 Relevance for the industry

Which directions the industry is going:
Some of the big companies in the industry that are creating IoT systems are currently pushing for solutions that include their customers in the actual building of experiences. An IoT life is simply too customized around each user, which makes a suitable experience difficult for anyone else but the user to design.

The conceptual method of building experiences we developed is not only bound to prototyping but could also be integrated into large systems creating real solutions. The concept might even be more suited. Systems like this would normally include a fixed collection of sensors and how they communicate, meaning users would not need to think about how the sensors communicate.

However, most of potential users of a system, like the one above might still be too technical for inexperienced user with its advance programming logic.

This makes the inexperienced users more insecure towards using the system. It would be a great feature for customers to be able to create customized experiences, but should never be a requirement. If a user is uncertain of how it would be done, there should always be an option to use already prepared flows.

At the same time we see initiatives for teaching children programming logic in an early age starting popping up. E.g. UK have started to include programming in their school curriculum as one of the first countries to do that [54]. This will hopefully lead to users not being as insecure when building an experience of this sort. Hopefully this will lead to a decrease in "trial and error" type of programming because the users know the basic logic of the systems.

Why our tool has an edge over existing tools:
The tool we designed focuses on decreasing the time it take from IoT idea to a first design iteration to actually test it. We also focused on taking away the technical knowledge normally required from the user.

There are a few tools that focus on the sensor integration as their main focus. One of them are the one create by Fogli et al [3]. However this tool is only focused on the social part (SMS, Facebook, twitter ect.) and cannot interact with a GUI for example. Other solutions such as the one made by Soukaras et al [2] offers the solutions to how the integration of the sensors could be simplified using their IoTSuite but does not take into account for the end users. It is more focused on the developers of prototyping tools. This is something that are quite common, most of the existing things only try to simplify the implementation of the sensors which would be the next step in this project.

There exists other tools that make sensor connectivity possible, E.g. Noodl or Node-Red, but they are either not initially developed for this purpose, or are to technically advance for Non Programmers to work with.

Which challenges of IoT prototyping still exists:
The greatest challenge for a tool designed for sensor connectivity is the vast variety of sensors and their different ways of communication with the tool. A tool like this is built on a technical foundation that constantly changes. This requires a living prototyping tool that constantly adapts to the market and user needs. This in turn creates high technical demands on the tool in terms of scalability and maintainability.

11.3 Project process evaluation

The work process was based on different methods that have lead up to our final results. The background research helped us understand strengths and weaknesses with existing tools and what they lack to be more viable with sensors. This also helped us to get more aligned with the industry which we had not been in contact with prior to this project. We started by researching the subject as wide as possible and as the project progressed narrowed it down to a higher level of detailed.

Each design iteration had its own method of evaluation according to the need of what we wanted to test. In the first iteration we wanted to get fast results and do fast changes where RITE was found suitable. The results from this method and these tests were successful and if we had to redo iteration one it would have been performed in a similar way.

The evaluation of our second iteration was more about confirming a concept where a prototype walkthrough was found suitable. The results of these tests might be a bit vague and we would have wanted to test on more users, especially users with no prior programming experience. The Hi-Fi prototype used during these tests might have been a bit too limited and restricted in functionality. This might have made the results a bit like we wanted them to be.

The changes in the third design iteration were evaluated with Nielsen and Molich's heuristic evaluation which we modified to also find solutions to the errors found within the tool. Two persons performed this evaluation, although three were recommended, which could have resulted in potential errors not being found. This is something we would have changed if we were to redo the project, to have external people do the heuristic evaluation and us only focusing on finding solution to the errors they find. In the fourth iteration no methods were used to validate the changes, this was due to the big scale of such a potential test.

Overall we feel that the methods used in this project were suitable and if we were to redo the project the same methods would have been used. When having a more or less clear goal of what you are working towards this iterative way of design works well. The possible changes we might would have done are the change of testing details stated in previous paragraphs. But since time is a factor it would be hard to e.g. test more persons or do a more thoroughly research.

Who did what during this project?
Joakim and Kristoffer have put equally amount of time into this project. Kristoffer has created all graphical visualizations of the tool. Apart from this, both Joakim and Kristoffer participated and performed equally in every phase of this research and design process.

11.4 Future work

Due to time limitation and the size of the tool we did not implement the final results of iteration four. We believe the size and complexity of the scenarios created in this version to be too difficult for an inexperienced user to perform within a reasonable testing time. To evaluate the functionality of the final results the tool would have to be developed and implemented to a level of completeness high enough to release it to ”real” users. By releasing an early functioning version of the tool software we can then gather information of what kind and size of User Experiences the users build. We would also experience which problems the users encounter and if the tool has unwanted limitations we did not expect it to have.

There are also graphical additions, e.g. more node icons in the menus to be created for a complete version of the tool.

If the concept proves successful after it is used by real users, the phase of building and supporting a community would need to follow.

11.5 Conclusions

The final prototype design of this project is a software tool design that lets users that have never programmed before create simple User Experiences that includes various sensors and multiple devices. These experiences can also be manually simulated in the tool without using physical sensors connected to the software.

Most of the results and conclusions of this report is not bound to software prototyping tools adapted to various sensors and multiple devices. The valuable tool features presented in chapter 5 are also of value to tools that does not include sensor integration. As well as the new method of building data flows.

References

[1] Söderlund T, editor. What is the best UX prototyping tool? [Blog post on the Internet]. Quora; 2015 [cited 2015 Jan 27]. Available from: http://www.quora.com/What-is-the-best-UX-prototyping-tool.

[2] Soukaras D, Partel P, Song H, Chaudhary S. IoTSuite: A ToolSuite for Prototyping Internet of Things Applications. 2015;.

[3] Fogli D, Giaccardi E, Acerbis A, Filisetti F. Physical Prototyping of Social Products Through End-User Development. End-User Development: 5th International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015 Proceedings. 2015;p. 217–222.

[4] Moen R, Norman C. Evolution of the PDCA Cycle;.

[5] Medlock MC, Wixon D, Terrano M, Romero R, Fulton B. Using the RITE method to improve products: A definition and a case study; 2002.

[6] Hundhausen C, Fairbrother D, Petre M. The "Prototype Walkthrough": A Studio-Based Learning Activity for the Next Generation of HCI Education. In: Next Generation of HCI and Education: CHI 2010 Workshop on UI Technologies and Educational Pedagogy. 2010;p. 2–4.

[7] Nielsen J, Molich R. Heuristic evaluation of user interfaces. Proceeding CHI '90 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 1990;p. 249–256.

[8] Who's Certified [Blog post from the Internet]. RFID Journal; 2009 [cited 20 Mar 9]. Available from: http://www.rfidjournal.com/articles/view?4986.

[9] Evans D. The Internet of Things - How the Next Evolution of the Internet Is Changing Everything. San Jose (CA): Cisco; 2011.

[10] What is Arduino? [homepage on the Internet]; c2012 - [cited 2015 Apr 23]. Available from: http://arduino.cc/en/Guide/Introduction.

[11] OCR Raspberry Pi Getting Started Tutorials [Tutorial pdf from the Internet]. OCR 2013 Oxford Cambridge and RSA Examinations; c2013 [cited 2015 Feb 2]. Available from: http://www.ocr.org.uk/Images/125879-getting-started-tutorials.pdf.

[12] Forum [Arduinos forum]. Arduino; 2015 [cited 2015 May]. Available from: http://forum.arduino.cc/.

[13] Forum [Raspberry Pis forum]. Raspberry Pi; 2015 [cited 2015 May]. Available from: https://www.raspberrypi.org/forums/.

[14] How it works [homepage on the Internet]. Austin, Texas: Supermechanical Limited Liability Company; c2012 - [cited 2015 Jan 20]. Available from: http://supermechanical.com/twine/features.html.

[15] Gartner's 2014 Hype Cycle for Emerging Technologies Maps the Journey to Digital Business [article on the Internet]. Gartner, Inc.; 2014 [cited 2015 Feb 19]. Available from: http://www.gartner.com/newsroom/id/2819918.

[16] Habib K, Leister W. Adaptive Security for the Internet of Things Reference Model. Norwegian Information Security Conference. 2013;p. 13–22.

[17] Zimmermann H. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. IEEE Transactions on communications. 1980;COM-28(4):429–31.

[18] Blessing B, Gibbins I, editors. Autonomic nervous system [Article on the Internet]. Scholarpedia; 2008 [cited 2015 Feb]. Available from: http://www.scholarpedia.org/article/Autonomic_nervous_system.

[19] Elkhodary A, Whittle J. A survey of approaches to adaptive application security; 2007.

[20] Minerva R, Berkers F. Use cases definitions and scenarios. ICT iCore; 2012. Project No.: ICT-2011-287708. Available from: http://www.iot-icore.eu/attachments/article/89/20120401_iCore_D11.pdf.

[21] 9 Real-Life Scenarios That Show How The Internet Of Things Could Transform Our Lives [Article on the Internet]. Business Insider Australia; 2014 [cited 2015 June]. Available from: http://www.businessinsider.com.au/9-real-life-scenarios-that-show-how-the-internet-of-things-could-transform-our-lives-2014-8.

[22] White RM. A Sensor Classification Scheme. Transactions on ultrasonic ferroelectrics, and frequency control. 1987;UFFC-34(2):2.

[23] Royce W. Managing the Development of Large Software Systems. Proceedings of IEEE WESCON 26. 1970;p. 1–9.

[24] Prakash HO, Phani Bhushan R, Varadan G, Venkataraman S. Integrated Visual Programming Environment. International Journal of Modeling and Optimization. 2013;3(3):256–8.

[25] Winberg M. Quotation from Noodl co-founder; 2015. Orally in a meeting.

[26] FORM SF 2014: Panel - Design Tooling [debate movie on the Internet]. Google Developers; 2014 [cited 2015 Feb]. Available from: https://www.youtube.com/watch?v=gX0n4JUl18g.

[27] Morrison JP. Flow-Based Programming. Journal of Application Developers' News. 2013;p. 1–5. Available from: http://ersaconf.org/ersa-adn/papers/adn003.pdf.

[28] Frei HP, Weller DL, Williams R. A graphics-based programming-support system. SIGGRAPH '78 Proceedings of the 5th annual conference on Computer graphics and interactive techniques. 1978;12,3:43–49.

[29] Ur B, McManus E, Pak Yong Ho M, Littmal ML. Practical Trigger-Action Programming in the Smart Home. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2014;p. 803–812.

[30] Maloney J, Resnick M, Rusk N, Silverman B, Eastmond E. The scratch programming language and environment. ACM Trans Comput Educ. 2010;10,4(16):1–14.

[31] Creating a Music Player [Tutorial from the Internet]. Malmö: Noodl AB; c2015 [cited 2015 Apr 16]. Available from: http://www.getnoodl.com/events.

[32] Introduction [Introduction page on the Internet]. Pixate, Inc.; c2015 [cited 2015 Apr 23]. Available from: http://help.pixate.com/knowledgebase/articles/461798-1-introduction.

[33] Pixate tool [Screenshot from video on the Internet]; 2015 [cited 2015 Apr 16]. Available from: http://www.pixate.com/education/demos/secret-comments/.

[34] Welcome to the InVision Learning Center [Tutorial guides from the Internet]; c2015 [cited 2015 Apr 15]. Available from: https://projects.invisionapp.com/d/main#/learn.

[35] Learning center [Learning tutorials from the Internet]; c2014 [cited 2015 Apr 15]. Available from: http://www.relativewave.com/form/learn/.

[36] Form tool [Screenshot from video on the Internet]; c2015 [cited 2015 Apr 20]. Available from: http://www.relativewave.com/form/learn/animations/.

[37] Node-RED A visual tool for wiring the Internet of Things [homepage on the Internet]; c2013 - [cited 2001 Feb 2]. Available from: http://nodered.org/.

[38] Node-RED tool [Screenshot from internet]; c2015 [cited 2015 Apr 20]. Available from: https://www.npmjs.com/package/node-red.

[39] About IFTTT [homepage on the Internet]; 2011 [cited 2015 Apr 23]. Available from: https://ifttt.com/wtf.

[40] Introduction [documentation on the Internet]; c2014 [cited 2015 Apr 15]. Available from: http://facebook.github.io/origami/documentation/.

[41] Introduction to Quartz Composer User Guide [User guide on the Internet]. Apple Inc.; c2004 - c2007 [cited 2015 Apr 15]. Available from: https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide/qc_intro/qc_intro.html#//apple_ref/doc/uid/TP40005381-CH201-TPXREF101.

[42] Origami tool [Screenshot from video on the Internet]; 2014 [cited 2015 Apr 15]. Available from: http://facebook.github.io/origami/tutorials/.

[43] Scratch tool [Screenshot from the Internet]; [cited 2015 Apr 15]. Available from: https://scratch.mit.edu/projects/editor/?tip_bar=getStarted.

[44] Pusch R. Alice: A Fresh Approach to Teaching Computer Science;. A 1-page handout giving a brief overview of Alice. Available from: http://www.aliceprogramming.net/overview/overview.html.

[45] Moskal R, Lurie D, Cooper S. Evaluating the Effectiveness of a New Instructional Approach. In Proceedings of 2004 SIGCSE Conference. 2004;p. 75–79.

[46] Alice tool [Screenshot from video on the Internet]; [cited 2015 Apr 15]. Available from: http://www.alice.org/index.php?page=aliceNewsAndNotes/aliceNewsAndNotes.

[47] Welcome to Bluetooth Technology 101 [homepage on the Internet]. Bluetooth SIG, Inc; c2014 [cited 2015 Feb 2]. Available from: http://www.bluetooth.com/Pages/Fast-Facts.aspx.

[48] Gislason D. Zigbee Wireless Networking. Newnes; 2008. Available from: http://www.eetimes.com/document.asp?doc_id=1278172.

[49] Hossain E, Leung K. Wireless Mesh Networks. Boston, MA: Springer Science+Business Media; 2008.

[50] Alljoyn System Description [homepage on the Internet]. AllSeen Alliance, Inc.; c2014 [cited 2015 Feb 2]. Available from: https://allseenalliance.org/developers/learn/core/system-description.

[51] SimpleLink™ Bluetooth Smart®/Multi-Standard SensorTag [product description on the Internet]. Texas Instruments Incorporated.; 2015 [cited 2015 Apr 16]. Available from: http://www.ti.com/tool/cc2650stk.

[52] Estimote API Documentation [Homepage on the Internet]. Estimote, inc.; 2015 [cited 2015 Apr]. Available from: http://estimote.com/api/.

[53] Medero S, editor. Paper Prototyping [Article on the Internet]. A List Apart; 2007 [cited 2015 May]. Available from: http://alistapart.com/article/paperprototyping.

[54] Cellan-Jones R, editor. A computing revolution in schools [Article on the Internet]. BBC; 2014 [cited 2015 May]. Available from: http://www.bbc.com/news/technology-29010511.

[55] Nielsen J, editor. 10 Usability Heuristics for User Interface Design [Article on the Internet]. Nielsen Noman Group; 1995 [cited 2015 Apr]. Available from: http://www.nngroup.com/articles/ten-usability-heuristics/.

Appendices

A Noodl exploration-prototype workspace description

The main screen, figure A.1, does not look that complicated, this is mainly because most of the “hard work” is done in other components. The group named Experience content is one component that looks simple but is a lot more to it. The nodes of Experience content can be showed in figure A.2, but all that it does is structuring another component which controls the different Experience items (highlighted in figure A.2). It is in the Experience item where the logical things happen. To explain every node in detail it would take a really long time so instead a brief summarization of the component would suffice for its purpose.

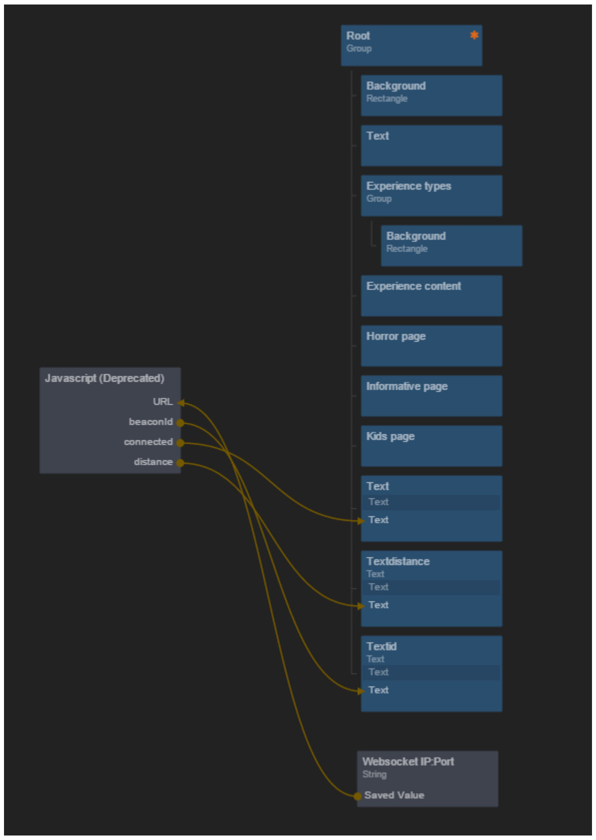


Figure A.1: The main page of the system.

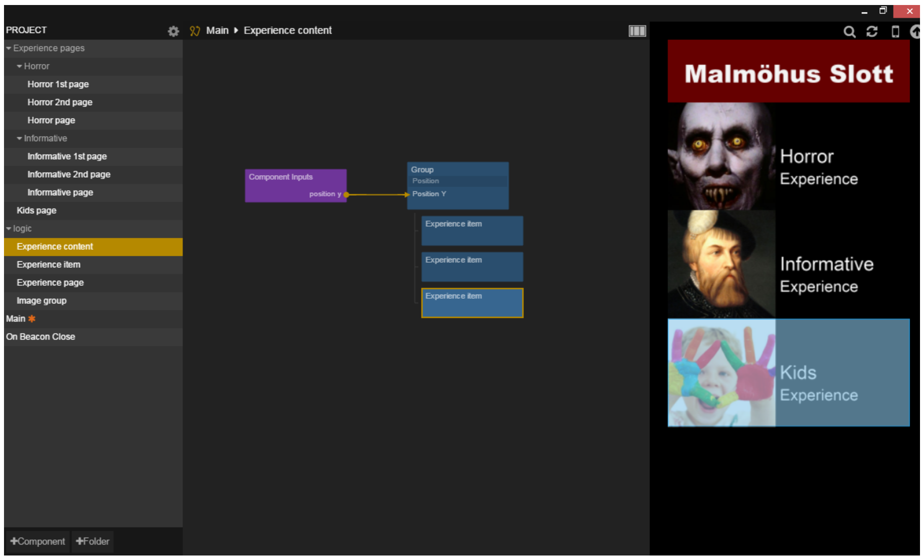


Figure A.2: A gather component to simplify other components. An Experience item is marked.

What the Experience item node (figure A.3) does is that when you swipe the component in the main page you will see a background color when sliding and the text "horror experience" for example fading away. It also sends an id to which experience that have been used so the next screen will show the right screen. When creating an Experience item component you have to add which background color when sliding the bar, which id the bar should have, which picture and what texts that you want to have included.

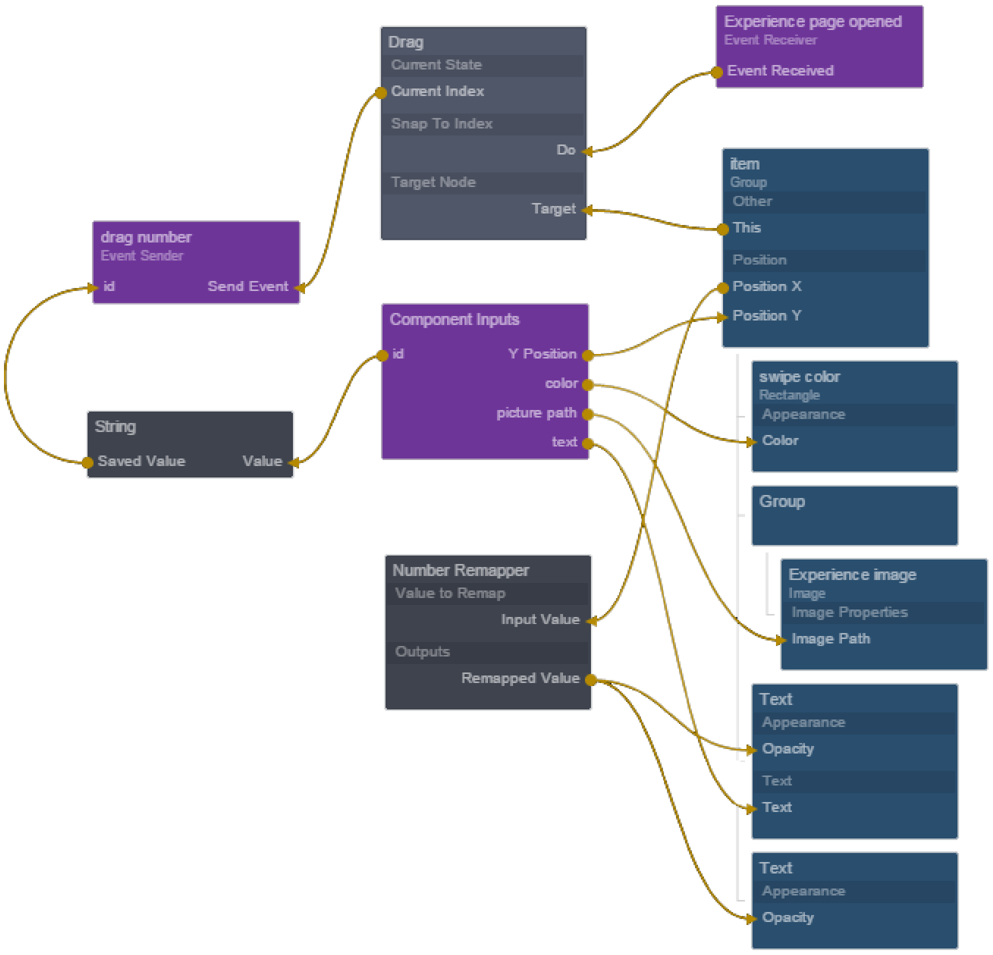


Figure A.3: The node flow of the Experience item.

Another more advance component that are used is the Experience page (figure 5.4). Basically what this does is that when you drag the Experience items that action and id is sent to this component. If it is the correct matching id the animation to the corresponding next page is made. It also creates a back button that can take you back to the main screen in an easy way.

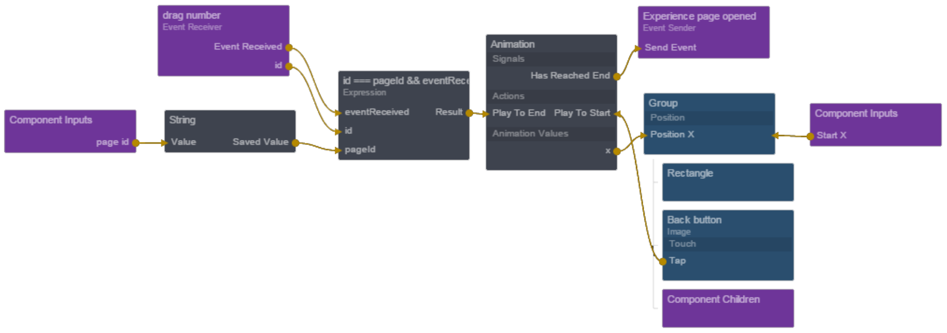


Figure A.4: A component which handles the changing of experience pages.

Figure A.5 shows the component that controls the handling of the information sent by the beacons and the phone. The key here is the JavaScript node; what it does is handling the string sent from our application, which is in JSON format, and parse out the parts we are interested in. In this component the beacon Id is saved and compared to the id of the experience you are in at the moment. This triggers the animation that changes the opacity of the page from 0, invisible, to 1, fully visible.

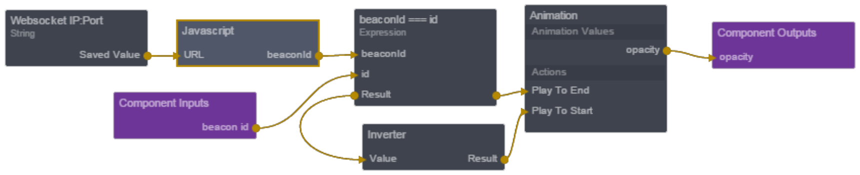


Figure A.5: The component that controls the information from the websocket.

B Nielsen's heuristics

These are the top ten most general principles for interaction design according to Jakob Nielsen [55].

Visibility of system status
The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world
The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User control and freedom
Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and standards
Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error prevention
Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Recognition rather than recall
Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use
Accelerators—unseen by the novice user—may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

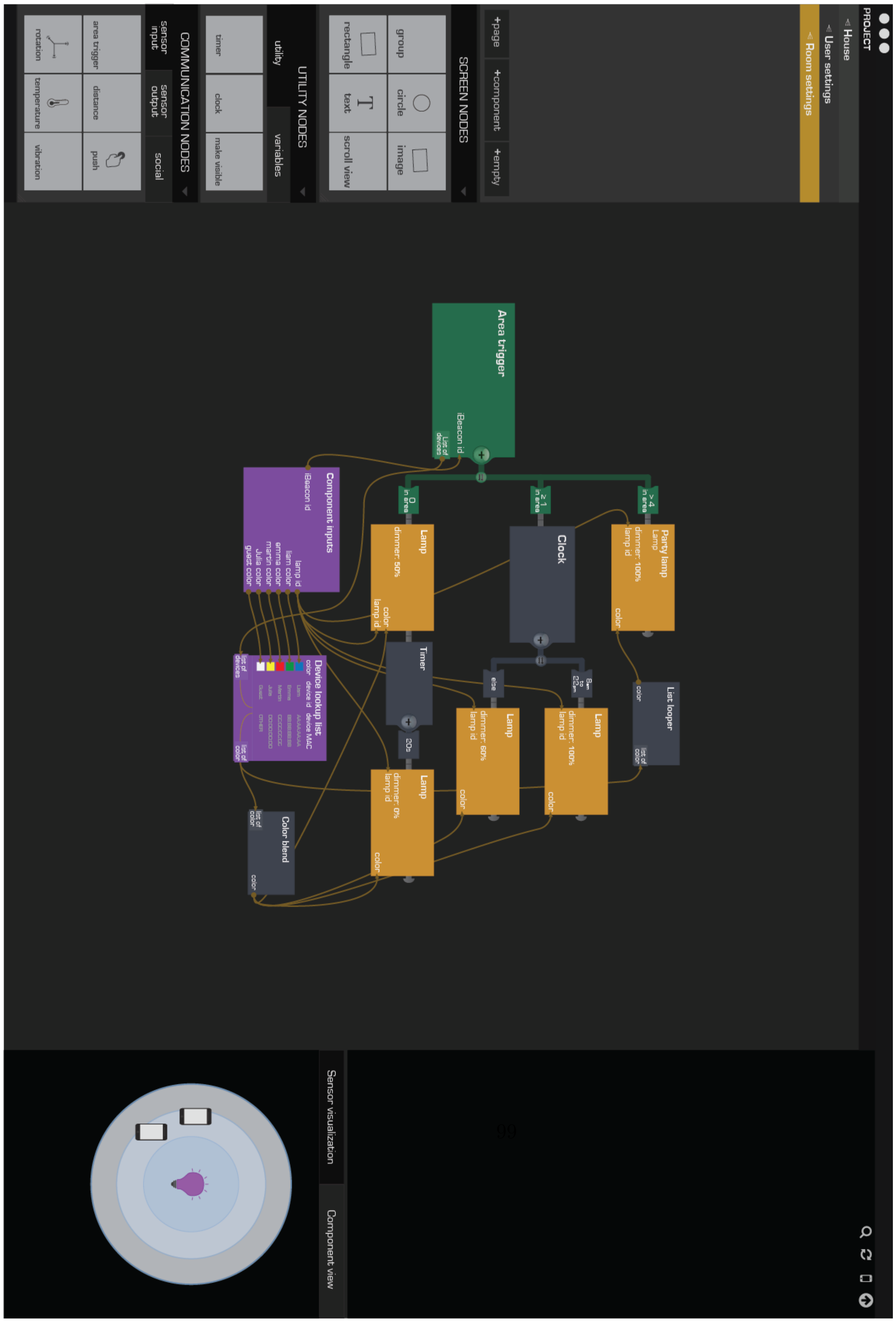
Aesthetic and minimalist design
Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large. ...

C Final tool prototype


PROJECT
House
User settings
Room settings
+page
+component
+empty
SCREEN NODES
group
circle
image
rectangle
text
scroll view
UTILITY NODES
utility
variables
timer
clock
make visible
COMMUNICATION NODES
sensor input
sensor output
social
area trigger
distance
push
rotation
temperature
vibration
Area trigger
iBeacon id
List of devices
0 in area
≥ 1 in area
> 4 in area
Component inputs
iBeacon id
lamp id
liam color
emma color
martin color
Julia color
guest color
Lamp
dimmer: 50%
color
lamp id
Clock
Party lamp
Lamp
dimmer: 100%
lamp id
color
Timer
20s
else
8am to 22pm
Lamp
dimmer: 60%
lamp id
color
Lamp
dimmer: 100%
lamp id
color
List looper
color
list of color
Device lookup list
color
device id
device MAC
list of devices
list of color
Lamp
dimmer: 0%
lamp id
color
Color blend
list of color
color
Sensor visualization
Component view