Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Magnus Bärnholt
Andreas S. Lyngby

An Internet-of-Things Software Framework for Exergames:

Architectural Requirements and Developer Acceptance

Master's thesis in Computer Science
Supervisor: Dag Svanæs
May 2019

NTNU
Norwegian University of Science and Technology

Magnus Bärnholt
Andreas S. Lyngby

An Internet-of-Things Software Framework for Exergames:

Architectural Requirements and Developer Acceptance

Master's thesis in Computer Science
Supervisor: Dag Svanæs
May 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Summary

Internet of Things offer a variety of solutions to connect sensors and integrated systems to the internet and expand the way we use the internet today.

This thesis is part of the EXACT research project where the aim is to explore physical exercise through exergames. The combination of exergames and internet of things is, as of now, a relatively unexplored field. This thesis will help create a platform that allows the creation of exergames.

In this thesis a framework was created for developing exergames using internet of things, and it has been evaluated and tested with game developers. The development of this framework has followed the principles of a Human User-Centered Design process, where the thesis completes one iteration.

The technology chosen is based on a set of criteria that was researched before the development of the framework started. These criteria fit the requirements of developing exergames using internet of things.

To realize the framework, two tile types were prototyped, each containing one Arduino based system on a chip, of the type ESP8266, with varying sensors and actuators fit for pervasive exergames. These sensors include a Time of Flight sensor and an inertial measurement unit. In addition, a Raspberry Pi working as an access point with an MQTT broker, was used as a communication layer between the devices.

The framework itself is partly targeting the Unity game engine. As such, it has been tested with Unity game developers with varying degrees of expertise, through usability testing.

The usability testing consisted of two workshops, where the participants first got to familiarize themselves with the framework before being given a task. The results of the workshop were based on System Usability Scale forms, code review, and semi-structured interviews. Based on the results of the tests, it was established that the criteria for the framework mostly covered the game developers' needs. However, there was a consensus that the frameworks should be seamlessly integrated into Unity, or a more Unity-based solution.

This study shows that game developers using the Unity game engine were interested in a framework that enables them to use internet of things in conjunction with regular game engines.

Sammendrag

Tingenes internett tilbyr en rekke løsninger for å koble sensorer og integrerte systemer til internett og utvide måten vi bruker internettet i dag.

Denne oppgaven er en del av forskingsprosjektet EXACT, der målet er å utforske fysisk trening gjennom exergames. Kombinasjonen av exergames og tingenes internett er et relativt uutforsket felt. Denne oppgaven vil bidra til å skape en plattform som gjør det mulig å utvikle exergames.

I denne oppgaven ble det utviklet et rammeverk for utvikling av exergames som bruker tingenes internett. Det har blitt evaluert og testet med spillutviklere. Utviklingen av dette rammeverket har fulgt prinsippene for en bruker-sentrert designprosess, hvor oppgaven har fullført én iterasjonen i en slik prosess.

Den valgte teknologien er basert på et sett av kriterier som ble undersøkt før utviklingen av rammeverket startet. Disse kriteriene passer til kravene for utvikling av exergames som benytter tingenes internett.

For å realisere rammeverket ble to typer 3D-printede fliser prototypet. Hver av flisene inneholder en Arduino-basert ESP8266 brikke, med varierende sensorer og aktuatorer som passer for gjennomgripende exergames. Disse sensorene inkluderer en Time of Flight-sensor og en 9-akse-bevegelsessensor. I tillegg ble en Raspberry Pi brukt som et kommunikasjonslag mellom enhetene, den var konfigurert som ett aksesspunkt med en MQTT-broker.

Rammeverket er delvis rettet mot spillmotoren Unity. Rammeverket har gjennomgått en brukbarhetstest der Unity-spillutviklere med varierende grad av erfaring har testet rammeverket.

Brukbarhetstesten besto av to workshops, der deltakerne først ble kjent med rammeverket før de ble tildelt en oppgave. Resultatene fra workshopene var basert på en brukbarhetsskala for et system (SUS), kodevurdering og semistrukturerte intervjuer. Basert på resultatene av testene ble det bevist at de gitte kriteriene for rammeverket hovedsakelig dekket spillutviklernes behov. Det var imidlertid enighet om at rammeverket burde være mer sømløst integrert imot spillmotoren Unity.

I denne studien kommer det frem at utviklere var interessert i et rammeverk der de kunne benytte tingenes internett til å utvikle nye gjennomgripende spill med bruk av Unity som spillmotor.

Acknowledgements

We want to thank Dag Svanæs for his supervision and his guidance throughout the entire project. He has helped clarify challenges during the implementation, his experience within usability testing has been very valuable, and we're especially thankful for his proof-reading of the thesis.

We want to thank Terje Røsand for his involvement in 3D printing of prototypes and advice regarding Arduino development. He has helped whenever we have had issues regarding the hardware, and if we ever needed any extra equipment he would always supply it.

We also want to thank both of them for their passion within their field of expertise which has been infections for us both.

Lastly, we want to thank the participants of the usability testing, who helped give us good feedback on the system they tested, and gave us an opportunity to test the framework, and Work Work for taking the time to let us present our project, it helped us recruit participants to the usability testing.

Magnus Bärnholt & Andreas Schatvet Lyngby
Trondheim May 31, 2019

Table of Contents

Summary i

Preface iii

Table of Contents viii

List of Figures x

Abbreviations xi

1 Introduction 1
- 1.1 Motivations . . . 1
- 1.2 Related work . . . 3
- 1.3 Objectives and Scope . . . 4
- 1.4 Research Questions . . . 4
- 1.5 Contribution . . . 4
- 1.6 Research Methods . . . 5
- 1.7 Participants . . . 5
- 1.8 Outline . . . 6

2 Backgound 9
- 2.1 Exergames . . . 9
- 2.2 Software Architecture . . . 12
  - 2.2.1 Software Architecture Patterns . . . 13
  - 2.2.2 Quality Attributes . . . 13
- 2.3 Internet of Things . . . 14
  - 2.3.1 Digital Twin . . . 18
  - 2.3.2 IoT Architecture Patterns . . . 21
- 2.4 Open Source Software . . . 24
- 2.5 Technology Acceptance Model . . . 26
- 2.6 User-Centered Design . . . 27

3 Research Methods and Design **29**
3.1 Research Methods . . . 29
3.1.1 Qualitative and quantitative research . . . 29
3.1.2 Interviews . . . 30
3.1.3 Prototyping . . . 30
3.1.4 Usability testing . . . 31
3.1.5 Guidelines for usability testing . . . 33
3.1.6 Questionnaires . . . 33
3.2 Validity of Research Methods . . . 34
3.3 Research Design . . . 35
3.3.1 Human-Centered Design: ISO 9241-210 . . . 35
3.3.2 RQ1: Requirements . . . 36
3.3.3 RQ2: Technology stack . . . 37
3.3.4 RQ3: Implementation . . . 37
3.3.5 RQ4: Usability and Usefulness . . . 38
3.3.6 RQ5: Updated Requirements . . . 39

4 Architectural Requirements **41**
4.1 Stakeholders . . . 41
4.1.1 Project team (Framework Developers) . . . 41
4.1.2 End-Users (Game Developers) . . . 41
4.2 Architecturally Significant Requirements (ASR's) . . . 42
4.2.1 Main functionality . . . 42
4.2.2 Criteria from Johansen . . . 42
4.2.3 Functional Requirements . . . 44
4.2.4 System Quality Attributes . . . 44
4.2.5 Quality Attribute Scenario . . . 45
4.2.6 COTS - Components and Technical Constraints . . . 47

5 Technology **49**
5.1 Technology Stack . . . 49
5.2 Hardware . . . 49
5.3 Software . . . 51
5.3.1 Connectivity . . . 51
5.3.2 Application . . . 51
5.4 Proposed Technology Stack . . . 52

6 Implementation **55**
6.1 Implementation Overview . . . 55
6.1.1 Component Roles . . . 56
6.2 Component Connection . . . 57
6.2.1 Arduino and Unity . . . 57
6.2.2 Unity and Digital Twin . . . 64
6.3 Device Configurations . . . 65

6.4 Examples . . . 67
6.4.1 Controller and Output . . . 67
6.4.2 Sample Tile . . . 67
6.4.3 Follow the Light . . . 68
6.5 Challenges and Reflection . . . 69

7 Usability Testing **71**
7.1 Research design . . . 71
7.2 Planning . . . 71
7.2.1 Recruiting participants . . . 71
7.2.2 Location and equipment . . . 72
7.2.3 User tests and tasks . . . 73
7.2.4 Semi-structured interview . . . 73
7.3 Procedure . . . 75
7.4 Results workshop 1 . . . 76
7.4.1 Observations: Problems and challenges . . . 76
7.4.2 Code analysis . . . 77
7.4.3 Group interviews . . . 77
7.4.4 System usability scale . . . 80
7.5 Results workshop 2 . . . 80
7.5.1 Observations: Problems and challenges . . . 80
7.5.2 Code analysis . . . 81
7.5.3 Group interviews . . . 81
7.5.4 System usability scale . . . 83

8 Requirements Update **85**
8.1 Summary of Semi-structured Interviews . . . 85
8.2 Updated Requirements . . . 86

9 Research Discussion **89**
9.1 Validity . . . 89
9.1.1 Objectivity . . . 89
9.1.2 External validity . . . 89
9.1.3 Ecological validity . . . 90
9.1.4 Triangulation . . . 90
9.1.5 Test subjects . . . 90
9.1.6 The task . . . 91
9.1.7 Researcher Bias . . . 91
9.2 Alternative Research Approaches . . . 92
9.2.1 Field Study . . . 92
9.2.2 Field Experiment . . . 92
9.2.3 Open Source . . . 92

10 Discussion **93**
10.1 Research Question 1 93
10.2 Research Question 2 93
10.3 Research Question 3 94
10.4 Research Question 4 95
10.5 Research Question 5 96

11 Conclusion **99**
11.1 Recommendations for Further Work 100

Bibliography **103**

Appendix **107**

List of Figures

1.1 Nintendo Wii workout, with a friend in multiplayer. 2
1.2 Playing with Moto Tiles 3
1.3 Oates's Research Process Model 5

2.1 AR, VR and Tangible Computing 11
2.2 Follow the Red Dot implementation from Johansen (2018) p. 23. 11
2.3 Simplified Technology Stack for IoT systems. 17
2.4 A simplified Broker Pattern diagram. 22
2.5 A simplified Multicast Pattern diagram. 23
2.6 A simplified Publish & Subscribe diagram. 23
2.7 Technology Acceptance Model. 27

3.1 Prototype triangle 31
3.2 Errors detected given the number of user tests 32
3.3 SUS response 34
3.4 Human-Centered Design 36
3.5 Model of research process for RQ4 38
3.6 Model of the research process. 39

4.1 Quality attribute scenario Bass et al. (2003) 45

5.1 3D printed prototype tiles, blue and red. 50
5.2 A modified version of the technology stack seen in chapter 2.2. 52
5.3 Proposed technology stack for this project. 53

6.1 Architecture Overview 56
6.2 Subscribe and Publish overview 58
6.3 Information flow overview 59
6.4 Sample Tile flow of events when tapped 68
6.5 Flow of Follow the Light example 69

7.1 Layout of the UX-lab with computers, tables and separator walls set up. . . . 73
7.2 Keep the Light Alive gameplay steps 74

11.1 Project.zip -> Arduino folder diagram 107
11.2 Project.zip -> Unity folder diagram 108

Abbreviations

Abbreviation	=	definition
AR	=	Augmented Reality
ARS	=	Architecturally Significant Requirement
AWS	=	Amazon Web Services
CAD	=	Computer-aided Design
CORBA	=	Common Object Request Broker Architecture
CoTS	=	Components and Technical Constraints
DHCP	=	Dynamic Host Configuration Protocol
DT	=	Digital Twin
DTPF	=	Digital Twin-driven Product Framework
EJB	=	Enterprise Java Beans
GPL	=	General Public License
HTML	=	Hypertext Markup Language
IMU	=	Inertial Measurement Unit
IoT	=	Internet of Things
ISO	=	International Organization for Standardization (from greek isos)
LED	=	Light-emitting Diode
MAC	=	Message Authentication Code
MQTT	=	Message Queuing Telemetry Transport
OSI	=	Open Source Initiative
OSS	=	Open Source Software
PLM	=	Product Lifecycle Management
QR	=	Quick Response
RFID	=	Radio-frequency identification
RPi	=	Raspberry Pi
SOA	=	Service-Oriented Architecture
SDK	=	Software Development Kit
SoC	=	System on a Chip
SUS	=	System Usability Scale
TAM	=	Technical Acceptance Model
ToF	=	Time of Flight
VR	=	Virtual Reality

Chapter 1

Introduction

1.1 Motivations

Exergames

Exergames are games that aim to improve and increase the user's physical activity, involving the use of large parts of the body. Wiemeyer and Kliem (2012) explain that such types of games have a lot to offer in physical activity and rehabilitation. The publication discusses whether exergames can be used to prevent and rehabilitate older people's health. It is evident from existing studies that exergames have a lot to offer, but only if the games are developed and designed based on an interdisciplinary understanding of the respective application fields. Figure 1.1 illustrates a workout game, played with Nintendo Wii's hand held game controllers.

> *Pervasive Games defines them as having "one or more salient features that expand the contractual magic circle of play spatially, temporally, or socially."*
>
> - Montola et al. (2009)

EXACT is an interdisciplinary research project between the Department of Computer Science (IDI) and the Department of Neuromedicine and Movement Science (INB) at NTNU. The primary objective of the project is to explore the use of exergames combined with social media in physical rehabilitation.

The secondary objectives of the project are: (I) developing a design methodology, and theoretical foundation for exergames in rehabilitation, (II) solving technical challenges related to applying the technology, and (III) evaluate the effect of exergames for rehabilitation. This thesis focuses the on secondary objective II, solving technical challenges.

Figure 1.1: Nintendo Wii workout, with a friend in multiplayer.

A publication from Konstantinidis et al. (2017) discusses how IoT(Internet of Things) devices such as game controls in the form of hand held devices or physical game objects in the environment are relevant for serious and pervasive games. They conclude that IoT devices are highly relevant in designing pervasive exergames.

A good example of a special kind of pervasive exergames is the Danish product Moto Tiles. Moto Tiles have embedded sensors and LEDs, and by placing them onto the floor, users can perform a variety of games and practices for fun, such as stepping the lighting tile and turning it off, making sounds by steps, etc. The use of Moto Tiles amongst elders in Denmark has shown remarkable effects of physical abilities in clinical effect studies, such as increasing balancing skills. Jessen and Lund (2016). The Moto Tiles in action is shown in figure 1.2.

Internet of Things

The Internet of Things gives us a world of new and interesting opportunities to create creative solutions in various applications. Everything from smart door locks, electric toothbrushes or complete surveillance of our house. Acknowledged cloud solutions like Amazon AWS, Microsoft Azure and Google Cloud, have all set their foot in the IoT. Today all of them are delivering a tailored ecosystem for IoT. In this thesis we want to develop a platform where IoT devices are connected to a WiFi network, and paired up with a digital twin in the

Figure 1.2: Playing with Moto Tiles

Unity game engine. They should communicate over a lightweight publish-subscribe-based messaging protocol, and have the opportunity to scale and communicate at a rapid speed. In order to realize such a platform, we need to find a comprehensive architecture that makes the platform stable and accessible to open source. It is also important that the architecture is scalable, forward-looking, safe and covers the quality attributes the platform should have. The motivation of this project comes from the possibilities IoT has been shown to offer, together with the wish of helping game developers to create more pervasive exergames.

1.2 Related work

To our knowledge there is not much research to be found on the need of an IoT platform, tailored for specific development of pervasive exergames. On the other hand, there is lots of research on why exergames is health beneficial, and why exergames is becoming more interesting in the field of rehabilitation and exercises of patients and older adults. Skjæret et al. (2016). As mentioned earlier, the Danish product Moto Tiles has been a source of inspiration for what exergames manage to achieve among the elderly. Moto Tiles has provided clinical evidence of efficacy and scientific articles regarding its product. Unlike what the EXACT project wants, Moto Tiles is developed with the idea of not being an open source product. Nor is it a general framework that anyone can use to develop their very own exergames. We have used the results and evaluation from the master's thesis of Johansen (2018) as a guideline to what criteria such an IoT platform may require.

1.3 Objectives and Scope

The main objective of this project is to develop an IoT platform for prototyping and developing pervasive games that utilize IoT technology based on open source platforms and frameworks. This platform will give developers less barriers of develop exergames, when using IoT devices as part of their game. The research is guided by five research questions presented in the following section.

As this is an exergames-focused project, we have chosen not to look at security in the communication between devices and games.

1.4 Research Questions

The overall research objective is to develop an open source IoT platform and find its requirements. The platform will undergo a usability test, and the uncovered errors will be used to update new requirements. The IoT platform will be a part of the EXACT research project, and its role is to make it easier for Unity game developers to create IoT-based pervasive exergames.

RQ1: What are the requirements for an open source architecture to support the development of pervasive exergames?

RQ2: What existing software technologies are best suited to realize these requirements?

RQ3: What are the challenges of implementing such an architecture for a specific target technology?

RQ4: How do potential users of the resulting framework asses its usefulness and usability?

RQ5: Based on the previous questions; what are the requirements for such an architecture?

1.5 Contribution

This paper will contribute to research done on IoT based pervasive game frameworks. The project will be to develop a platform where IoT devices can be connected as a digital twin to an already existing game development platform. In order to realize such a platform, we need to find a comprehensive architecture that takes the findings from Johansen's thesis Johansen (2018), and build upon them to make the platform stable, secure and accessible with open source in mind. Undefined aspects like overall performance and scalability will be measured and discussed along the way. The platform will undergo a usability test, where we invite potential game developers with different level of programming skills to evaluate the design. We hope that the results of this project will help game developers with different levels of competence to easier develop pervasive exergames. If the platform evolves in the open source community, we believe that more exergames will see tomorrow's light and help people through rehabilitation or give them a new way of playful exercise.

1.6 Research Methods

Our research methods are based upon the literature from the book *Researching information systems and computing* by Oates. We will define our research questions based on a mixture of conclusions drawn from Johansen's master thesis and a literature review. Afterwards, we will go through a design and creation process where the framework will be developed. Finally, an experiment will be conducted, where a usability test will be carried out on the framework. Oates emphasizes that research projects should demonstrate not just technical skills, but also academic qualities, for example analysis, argumentation and critical evaluation, so it creates some new knowledge. Oates (2012). So, for our master thesis we will round up the report with a research discussion, where we reflects upon the findings and the validity of the experiment. Figure 1.3 outlines our research plan based on Oates' research process model.

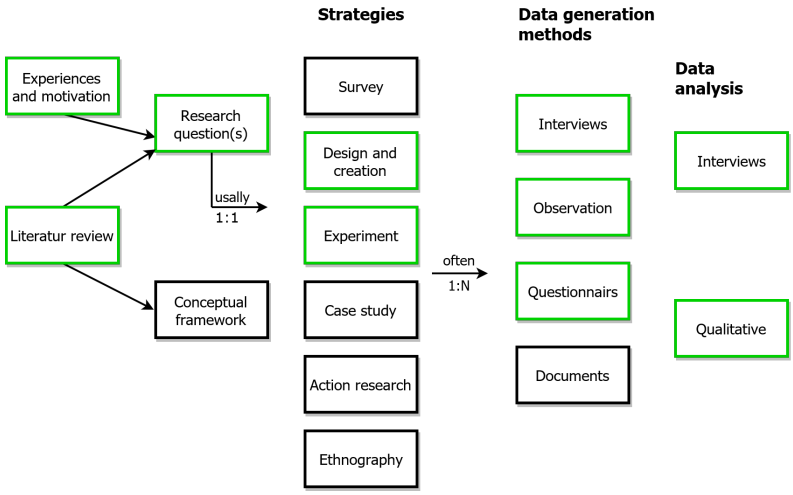


Figure 1.3: Oates's Research Process Model

1.7 Participants

The research was conducted by: Andreas Schatvet Lyngby (student), Magnus Bärnholt (student), Dag Svanæs (professor) and Terje Røsand (senior engineer). Terje provided us with technical advice regarding the Arduino hardware with its sensors. He was also responsible of the hardware design and 3D-printing the prototypes. Dag was the lead supervisor who would frequently monitoring our research progress and helped our focus to be pointed in the right direction. For this master thesis, we planned to include some experiments on a selective group of people. This project has NSF(ethics) approval as part of the EXACT project. Users had to sign informed consent forms, and we made sure the data was stored on NTNU's data servers. We attached great emphasis in recruiting participants, and were planning a presentation of the project for developers at the incubator Work Work in Trond-

heim. We wanted to recruit game developers who had both high competence and students with less experience. This would give us the opportunity to make a broader analysis of the evaluation results. Under the experimental stage of the research we wanted to participate under guidance of Dag.

1.8 Outline

Chapter 1

Has an introduction to the research being done, along with some background information for motivation and related work, and research questions to be looked at. Sets the baseline for the report, and contains this outline for the rest of the report.

Chapter 2

Chapter 2 aims to explore some of the themes relevant to the research. Going over some relevant background information with topics that fit the research. These topics include software architecture, internet of things, digital twin, open source software, exergames and lastly user-centered design and TAM.

Chapter 3

This chapter details the research methods and the validity of them, and ends with covering all five research questions, giving them a more detailed explanation.

Chapter 4

This chapter covers the documentation in more detail in regards to the architecture. This is more of an overview over the architecture itself, as well as some criteria set for ourselves in terms of development of the architecture for the system.

Chapter 5

In chapter 5 we explore the technologies we used in our research, and in the design and creation phase of the project. This includes software and to some extend hardware, and also the technology stack suitable for our aimed product.

Chapter 6

In chapter 6 we detail the implementation at its current state. This includes an overview of the architecture, in the way the components described in chapter 5 are connected, and also how the software side works. The software code is also explained to some extent. In addition, we go over some examples and some challenges during the implementation.

Chapter 7

This chapter details the entire usability testing, which includes research design, planning, procedure and the results for the workshops.

Chapter 8

In chapter 8 we briefly cover some changes to the requirements that surfaced during the usability testing.

Chapter 9

In this chapter we discuss the validity of our research, and explore alternative research approaches.

Chapter 10

In this chapter we give a deeper discussion on each research question detailed in chapter 1 based on the results from our research approach.

Chapter 11

This chapter concludes our research for each research question. This chapter also has some recommendations for further work.

Appendix

The appendix contains two folder structure diagram, documentation on the framework, a questionnaire, SUS form, and lastly the partner agreement.

Chapter 2

Backgound

2.1 Exergames

Over the last decade, a lot of attention from the research community has been devoted to some of the growing health-related issues in the modern western society. Some of these issues relate to the increasing need of physical activity and improvements of health and physical function in older adults Skjæret et al. (2016), as well as the growing problem of obesity in the younger demographic due to less physical activity and non-physical activities related to playing video games and watching television Oh and Yang (2010). Some of these issues stem from the way modern technology has transformed our lives, making us comfortable and more prone to sedentary lifestyles. Given this, a focus has been put on exploring a new type of games referred to as exergames, or exercise-games, and exergaming. These are a subset of serious games, games more focused on mental and physical benefits, not just entertainment.

The term exergames and exergaming has been given various definition and had various types of use due to interests spanning multiple research collectives. Oh and Yang (2010) has done a review on the term usage within both health related and non-health related research to propose a new definition that combines exertion and video games. This type of exertion includes strength training, balance and flexibility. From what they found, most researchers consider exercise differently and has used the term in different settings. Some consider all type of non-sedentary activity as exercise, while others have standardized methods of monitoring activity levels. This makes comparing the different studies difficult. To standardize the use of exergames and exergaming, Oh and Yang (2010) proposes the following definitions, these are also the definitions we will follow:

Exergame A video game that promotes (either via using or requiring) players' physical movements (exertion) that is generally more than sedentary and includes strength, balance, and flexibility activities.

Exergaming An experiential activity where playing exergames, video-games, or computer-based games is used to promote physical activity that is more than sedentary activities and also includes strength, balance, and flexibility activities.

Benefits of exercise gaming

The benefits of exergames comes when you put your body in motion. Several exergames have physical benefits similar to aerobic exercise, and helps reduce the risk of heart and vascular disease, high blood pressure, obesity and diabetes. Manley (1996). In a report from Siegel et al. (2009), scientists show that playing 30 minutes of exergames results in an increased heart rate and calorie consumption that's within what the American College of Sports Medicine recommends for daily physical activity.

AR, VR and Tangible computing

When we talk about pervasive exergames, three new technologies are often discussed. These are augmented and virtual reality, as well as tangible computing. Figure 2.1 provides a graphic illustration of these three technologies. The augmented reality means that reality is combined with the digital. Augmented reality can be created through e.g. a mobile screen, where the camera observes the surrounding environment, and software generates data images that is placed over the image of the physical world. Virtual reality on the other hand generates a picture of a digital reality, and then locks you into this reality by an isolated screen. One of the industries that has truly embraced augmented reality is the gaming industry. A good example of a exergame using AR technology is Pokemon GO. In our thesis, we will use physical objects to manipulate the digital state of a Unity gaming object, which falls under the tangible computing category. The definition of a tangible user interface is:

> *A tangible user interface can be defined as a device that converts physical interactions into digital information and vice versa; it can therefore be considered as a translator between the real and virtual environment.*
>
> - Patten et al. (2000)

Figure 2.1: AR, VR and Tangible Computing

Examples of Exergames

In his thesis, Johansen (2018) details a proof of concept for an exercise game using micro-controllers connected to a game engine called Unity. Here, he explores the use of different technologies that can be used to connect a game engine and IoT technologies. From his research spawned a sample game "Follow the Red Dot" in which the micro-controllers light up and the player(the user of the system) has to touch the micro-controllers to turn the light off. Another light will then appear on a different micro-controller. This proof of concept also sets the starting point for our project. In figure 2.2 we see an implementation prototype of this concept where the dot moves from one place to another.

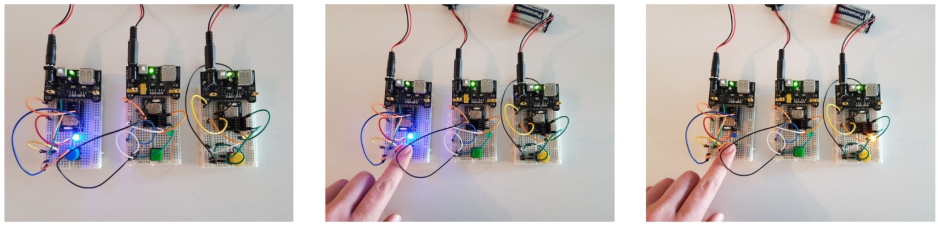


Figure 2.2: Follow the Red Dot implementation from Johansen (2018) p. 23.

Another example of exercise games that follows the same concept direction is from Moto (2017), called Moto Tiles(recall from chapter 1). This is a more complete package and a finished product of what Johansen (2018) explored in his thesis. With an Android

tablet and their IoT tiles with pressure sensitivity, they have several modes of play where the players either touch or step on the tiles to use them as controllers. See figure 1.2 from chapter 1.

IoT-based Exergame Platforms

There are existing IoT-based frameworks and platform for games and exergames. Some of these frameworks do however not have names, but they do have example games listed.

Henry (2018) has an interesting framework that uses MQTT as the middle-ware with a Broker Pattern. They do have an example application called SEA: Student Engagement Application. This one is interesting for reason we'll get into in later chapters of this report.

Konstantinidis et al. (2015) has their framework for exergames removed from the traditional desktop application by embedding the application into HTML5 and thus bringing the use of mobile devices for full body exergames.

Martin and Laviola (2016) brings the Transreality Interaction Platform which is a platform where they mix Virtual Reality components and IoT.

Among other frameworks and platforms we have UKKO(Dickinson et al. (2015)), a location-based pervasive game for students. There's also the InLife (Kosmides et al. (2018)) framework for IoT-based Serious Games.

2.2 Software Architecture

Bass, Clements and Kazman define software architecture the following way:

> *The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.*
>
> - Bass et al. (2003)

Important to note is that this definition doesn't limit itself to only define software architecture in a way where the architectural decisions are made early. With agile or spiral-development, as examples, these decisions are made not only early, but also later in development and implementation. Essentially, software architecture is all that describes the system, from the implementation itself to the various types of documentation surrounding it. For example, this report itself along the code and implementation, which will be explained in further detail in chapter 4.

Within software architecture you have a few categories that help describe the system. These are structures, views and patterns. A structure is the set of elements in the system, as they are in the software or hardware. Views are representations of a coherent set of architectural elements within the system as a whole, as it's written and read by stakeholders, thus also the relations between the elements. Patterns are strategies on how to solve the problems the system will aim at, patterns are elaborated in the subsection 2.2.1.

Shortly explained, a view is a representation of a structure. Meaning if you have what's called a module structure, the model *view* is the documented representational view of said

structure, which is made according to a template and used by stakeholders. Structures on the other hand come in three major categories we will cover in short Bass et al. (2003)

- **Module structures:** These are essentially which decisions are used in how the system is to be structured as a set of code or what kinds of data units have to be constructed or procured. These are static representations of the system, and have areas of functional responsibility.
- **Component-and-connector structures:** These are decisions as to the system's structure of the run time behaviors(components) and interactions(connectors). The views of this kind of structure are important in questioning the system's run time properties, such as performance, security, availability, etc.
- **Allocation structures:** These are structure that show the relationship between software elements and other external environments in which the software is created and executed. Often structured around deployment, implementation and work assignment.

2.2.1 Software Architecture Patterns

Intel's web-article on communication patterns briefly mentions that it's important to decide on an architecture pattern early, so you don't paint yourself into a corner. Intel (2016). This is mostly in terms of a project using devices and communication protocols, much like our own. Bass, Clements and Kazman, defines architecture patterns as following:

> *An architectural pattern is a package of design decisions that is found repeatedly in practice, has known properties that permit reuse, and describes a class of architectures.*
>
> - Bass et al. (2003)

The patterns themselves establish a relationship between various element; A context, problem, solution, and element types, interaction mechanisms or connectors, topological layout of of the components, and various semantic constraints. This relationship works as a way to reason about the problem at hand. Which will be described throughout this report.

2.2.2 Quality Attributes

Quality attributes are the criteria that we define as non-functional requirements. These are requirements used to evaluate the system as a whole, but not for specific behavior. Quality attributes can be divided into two main categories. The attributes that affect system behavior, design and the user interface, and those that affect the development and support of the system. To achieve the greatest success with the architecture, it is a good idea to combine quality attributes such as performance, reliability and ease of use. It is also important to consider the impact the characteristics may have on the functional requirements and weigh these against each other.

2.3 Internet of Things

In an article in the RFID Journal released in 2009, Kevin AshtonAshton et al. (2009) claims to have coined the term "Internet of Things" back in 1999 when he had a presentation at Procter & Gamble. His idea was for them to implement RFID into their supply chain.

What he believed back then and still believed at the time he wrote that article in the RFID Journal, was that computers at the time, and today as well, relied heavily on human beings for their information. Any type of data on the internet has been mostly built up by humans in the way of interaction with the computers and the internet. The problem here lies in the fact that this is not a very reliable source for information about the real world. This is where the “things” come into play.

Since humans and the environment humans live in are physical, they’re in essence based on things. Humans need the physical to live. This makes the “things” that much more important compared to the ideas and information the internet represents without the “things”. This also means that IT at the time of 2009, was so dependent on data from people, and knew very little about the physical.

However, this is where the things he spoke of comes into play. If the computers knew everything there was to know about things, with data gathered without the direct input from humans, then this additional data could help us in many different ways. This is why he thinks computers need their own way of gathering information. Which in turn will help the computers “see, hear and smell” the world themselves.

This is where the RFID and sensor technology comes in. This technology would be there to help computers get all this information. To then observe, identify and understand the world, without depending too much on human input.

Early stages of IoT

Tracing back even before Ashton coined the term, the concept of IoT is old, more than 25 years old. Depending on how you choose to look at the history of IoT, what started it was arguably a webcam pointing at a coffee machine. BCS (2017). This allowed for a stream service from the camera that could be accessed in a web browser, and was considered revolutionary in 1991. However, in the media it was considered a gimmick and was at a relative level dismissed by the majority as they couldn’t see any better application for the webcams, which are at present day so commonplace and no longer considered part of the IoT ecosystem.

Development continued beyond this and RFID tags were starting to get used in development. In Prof. Sanjay Sarma’s research team BCS (2017), one of the professor’s students combined a simple RFID tag reader with a microwave oven. His idea was to have the oven read tags connected to food items, then download the needed cooking instructions and thus cook the item without needing any further human input.

From Sarma’s group, RFID became a huge focus and the Electronic Product Code movement was started. With this, RFID as a product and utility was brought to the market some years later. Though, it didn’t take off properly before the economic crash in 2008 where companies realized they needed to update their processes with inventory and prod-

ucts. Through development by looking at different technology area, the cost of the sensors were brought down and spread more.

However, today RFID technology is only a subset of what IoT really is, and the scope of what IoT really is has been further developed and extended way beyond just RFID or "extended barcodes". There are more sensors and actuators, and other hardware, along with software that builds up what we know as IoT now.

Directions of IoT

In general, IoT is a very broad term and it's relatively difficult to properly define what it is, along with the understanding of its scope. There is a suggested separation of IoT's development where you have two paths; the retrospective and prospective.

In the retrospective direction of IoT the main focus is on the uniqueness of things online. The uniqueness here defined by the devices themselves usually only have a one-way communicative connection to the internet, or to some other cloud. This is often achieved through RFID and QR codes. This direction of IoT development is also mainly defined in the category of hobbyist or private projects, though there are some companies that have taken this direction of IoT development into use. An example of this is Apple's wireless beacons to send messages to phones. This direction does unfortunately not have much of a commercial use and there has only been few major interconnected IoT projects within this category.

In the prospective direction of IoT the main focus is having the thing as a complete package. Think of it as a readily available product for the consumers, like a fridge or a toaster, or something similar. Here the device and the cloud has bidirectional communication connectivity. The communication is embedded directly into the core of the thing and the thing's functionality is developed with this in mind. The general populace is still trying to figure out the benefits of having various things connected to the internet, but TVs are getting smarter and smarter and could be considered IoT in its own right, even if they're moving more and more towards being complete packages of a computer.

Further Defining IoT

When it comes to defining what IoT is, or what kind of ramifications IoT has, there are many suggested definitions. Especially, also, considering what IoT entails. The International Telecommunication Union puts it nicely:

> *A global infrastructure for the Information Society, enabling advanced services by interconnecting(physical and virtual) things based on, existing and evolving, inter-operable information and communication technologies.*
>
> - Wortmann and Flüchter (2015)

This definition is particularly nice because it encompasses something relatively close to a full definition covering most aspects of IoT. The things, both physical and digital. The continued development. The communication between the things. How interconnected these things become with those interacting with them.

Smart Industry	Intelligent production systems Connected production sites Industry 4.0
Smart Home	Intelligent thermostats Security systems
Smart Energy	Smart electricity Gas and water meters
Smart Transport	Vehicle fleet tracking Mobile ticketing
Smart Health	Patient surveillance Chronic disease management
Smart City	Real-time monitoring of parking space availability Intelligent lighting of streets

Table 2.1: List of areas of IoT and some applications

There are also other definitions for IoT. Some of which focus more on the things themselves, and their connection to IoT. Some of which focus more on the internet-related aspects of IoT, meaning how the things interact, with internet protocols and network technology. And some of which are more related to the semantics challenges of IoT, covering storage, search and organization of large volumes of information. Though these are some focuses that definitions often fall under, there are also other definitions for IoT that are more similar to the one from the International Communications Union.

Applications of IoT

There are many diverse applications for IoT, and the solutions based on IoT seem to be increasing in virtually every aspect of everyday life. In table 2.1 some areas of IoT are covered along with a couple of applications-areas for each.

All of these areas are the result of innovation of IoT. This innovation has, through combining the physical and digital, created a vast array of new products, as well as enabled new novel business models. With IoT, new technology is also developed at the same time as the technology behind the IoT products gets further developed, increasing the quality of various components used within IoT. A lot of this new technology has also helped in making digitizing the things easier.

IoT has also brought new opportunities for incremental value generation. What this means is that a physical thing is enhanced with the digital in newer and newer ways. First on a local level and then as they are further developed, on a global level.

An example of this incremental value generation is a simple light bulb. Its primary function is lighting up areas. With IoT and various sensors, the light bulb can detect human presence and in turn act as a low-cost security system if it blinks and sends a message to whoever owns the light bulb.

Another example is by taking a simple storage bin. Its primary function is storage. With IoT you can have sensors that detect how many items are in this storage and can then

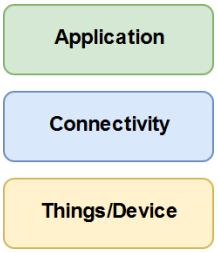


Figure 2.3: Simplified Technology Stack for IoT systems.

detect low stock. If it's connected further, it can also send out requests to be replenished.

Innovation within IoT is, however, not only limited to the individual connected thing. This innovation can be brought over to a multi-product layer in terms of the products being connected as part of a product system. An example of this could be a tractor connected to a larger ecosystem for farm equipment including other vehicles at the farm. This then used to monitor the different equipment and measure the efficiency of the equipment in the system. To take it even further, multiple product systems could be interconnected. This type of connection of systems in systems can expand industry boundaries.

Technology Stack in IoT

From a technological standpoint, the implementation of the software and hardware used in IoT product solutions usually come in a multi-layer technology stack where you have three main categories of layers. The thing or device layer, the connectivity layer and the cloud layer.

At the thing or device layer, you usually have your typical IoT-related hardware, in some cases Arduinos, with additional components such as sensors, actuators and even in some cases more processors components. Along the hardware is usually also some embedded software which can be modified or integrated to manage and operate the physical thing.

At the connectivity layer you have communication protocols such as MQTT or WebSockets which aids in the communication between the thing/device and the last layer, the cloud layer.

At the cloud layer you have various management software which helps in device communication, and is used to communicate with, provision and manage the various connected things. At this level is also an application platform which helps in terms of development and execution of IoT applications. A lot of the management also comes down to analysis and data management, while there can also be a process management software which helps to define, execute and monitor processes between the categories of people, system and things.

In figure 2.3, a simple technology stack of the various layers can be seen. A more detailed overview of a suggested full technology stack, with more details of what goes into each layer, can be seen in the article Technology and Value Added by Wortmann and Flüchter (2015).

In the more detailed technology stack diagram, they have listed a few more elements that span the three layers in the technology stack. The first is something they call Identity and Security, which covers everything of user authentication and access to assure security across the different layers. The second one being Integration with Business Systems, which covers software components used to integrate enterprise systems. Lastly we have External Information Sources, which covers software components to enable integration of external, third-party information.

2.3.1 Digital Twin

The concept of Digital Twin is a relatively recent one, and there's a bit of a split exactly where the terminology and concept of it stems from. According a paper on the subject from Glaessgen and Stargel (2012), NASA and the U.S Air Force, the term Digital Twin was first conceived at DARPA, and then later Grieves and Vickers (2017) properly conceptualized the terminology around it in their early works of Product Life cycle Management(PLM) from 2002 until around 2011 where the concept was finalized to where it is today. However, according to Tao et al. (2018) the most commonly used definition of Digital Twin was proposed by Glaessgen and Stargel:

> *Digital twin means an integrated multiphysics, multiscale, probabilistic simulation of a complex product, which functions to mirror the life of its corresponding twin.*
>
> - Glaessgen and Stargel (2012)

Before the concept of Digital Twin was properly in place, the product development of complex systems were at a completely different scale. With only being able to work with physical objects, the range of the investigation done during testing was only limited to when they could have the product in a physical environment with actual forces. This meant the first time a condition not covered by the physical tests would show up, would be when the object was in actual use. This increased the unpredictability of the product, as well as cost and time-consumption in fixing these errors.

According to Grieves and Vickers (2017) there are three types of systems; simple, complicated, and complex. Simple systems are simple by nature because it's predictable and the observer can easily understand how it works. Complicated systems are also predictable. However, a complicated system is comprised of a larger number of components, but it's still predictable and straightforward, Complex systems however are characterized as large networks of components with many-to-many communication channels and advanced information processing. These complex systems are complex because they're not predictable like the two others and results are difficult to discern.

The issues appearing when just testing on the physical, and the complex systems, are what the concept of Digital Twin is aimed at combating. Looking back at the previously mentioned PLM, this is where the digital twin had its groundwork established. The idea for PLM comprised of elements that now make up what's known as a Digital Twin; a physical space and a virtual space. Between these two spaces, there would be a link for data to flow

both directions. This gives us a system that consists of the two systems with the physical being as it always has been, and the virtual space containing all information available about the physical. This gives a mirrored system. Grieves and Vickers (2017) has proposed four extra definitions when it comes to Digital Twin and it's different manifestations. These four are the following:

- **Digital Twin:** A set of virtual information that describes a potential or actual physical object or product ranging from micro to macro level. At its core, any information that could be gotten from the actual physical object should be obtainable from the virtual object as well. This is the "parent" definition for the three other definitions, which include the Digital Twin Prototype, Digital Twin Instance and Digital Twin environment.
- **Digital Twin Prototype:** This is a type of Digital Twin that describes a physical prototype object. Meaning it has the information required to describe and produce a physical version of the virtual object. This is akin to a class in terms of programming.
- **Digital Twin Instance:** This is a type of Digital Twin that describes a specific corresponding physical object. This is on an individual level, and means the link between the virtual and the physical remains throughout the life of that physical product. This is more akin to an instance of a class.
- **Digital Twin Environment:** This is the integrated, multi-domain physics application space for operating on Digital Twins for a variety of purposes.

In essence, the Digital Twin concept is about information. The information used in the Digital Twin model is there to replace the wasted physical resources that was a problem in the past before the digitization which happened at the end of the 20th or start of the 21st century.

A good example of this information use is the manufacturing of large vehicles that NASA and the U.S. Air Force has detailed from Glaessgen and Stargel (2012). The vehicles produced, and the problems they face, are that of complex systems where, in NASA's case, they have an extensive set of requirements and regulations in place for how a vehicle should operate, and their required performance. With the help of digital twins, there has been put systems in place on the vehicles where the Digital Twin can forecast the health of the vehicles or system, its remaining life and probability of mission success. With the help of the Digital Twin, a prediction can also be made on the safety of critical events, and help predict unknown issues before they become critical. The systems should also be able to mitigate damage or degradation with the help of self-healing mechanisms or recommend changes in the vehicle's mission profile to increase life span and mission success probability.

While Grieves and Vickers (2017) and Glaessgen and Stargel (2012) has focused more on the heavy-duty, complex system side of things for the Digital Twin paradigm, Tao et al. (2018) claims that little had been done in terms of exploring the value of DT in terms of consumer product design. With this, they have looked at various different data-driven design frameworks for product development and concluded the need for a Digital Twin-driven Product Framework, to help generalize and set a methodology for the development of Digital Twin-based products.

With their Digital Twin-driven framework, Tao et al. (2018) proposes three parts that the Digital Twin mode consists of. The first part are the physical entities in the physical space. These are real products that can be operated by the users, man manufactured from raw material or parts. During it's life cycle, these physical entities have varying characteristics, behaviors and performance.

The second part of the Digital Twin mode is the virtual space. These are virtual mappings of the physical products in virtual space. Preferably, they reflect their physical twin throughout the whole life cycle, and also simulate, monitor, diagnose, predict and control the states and behaviors of their physical twin. These virtual models should also preferably have information about the geometric models of the physical entity, as well as its rules and behaviors, with material properties, mechanical analysis, and health monitoring. As well as various other information about the physical object. While these are preferable, the twin can also be simpler with less data about the physical entity, but enough to mirror it properly with behavior and functionality.

The third part of the Digital Twin model is the connected data that ties the physical and the virtual spaces. This data includes the subsets of physical and virtual data, as well as new data that is acquired after integration, fusion and analysis of the physical and virtual data. The purpose of this third part is to achieve a two-way data transmission process between the physical and virtual. This gives us a closed-loop process.

In their creation of the Digital Twin-driven Product Framework, Tao et al. (2018) has proposed six steps needed to create a fully functional Digital Twin. However, while these are considered steps to creating Digital Twins, there is no need to strictly follow the steps in sequential order, and can be done simultaneously. The steps in order to create a fully functional Digital Twin according to Tao et al. (2018) are as follows:

1. **Build the virtual representation of the physical product**
   This step is usually done with computer-aided design (CAD) and 3D modelling. This virtual product usually includes three parts; elements, behaviours, and rules. The elements usually means the geometric model and physical model together. The behaviours usually mean not only the behavior of user and product, but also analysis of of the product and user interaction generated by this behavior and modeling. The rules are usually the evaluation, optimization and forecasting models.

2. **Process data to facilitate design decision-making**
   Data collected from various sources are analyzed, integrated and visualized. Raw data is converted into concrete information which helps designers in decision-making about the product. The data being from various sources also helps discovering patterns that wouldn't be discovered if the source was singular. Visualization is used to present the data in a meaningful way. In this step, artificial intelligence can also be incorporated to help enhance the DT's cognitive ability, so it can make some simple recommendations itself.

3. **Simulate product behaviours in the virtual environment**
   This step is helped by simulation and virtual reality. The simulation can help simulate

how the product would behave in the real world. Virtual reality can be used to help involve designers and creators interact directly with the virtual product.

4. **Command the physical product to perform recommended behaviours**
   Based on recommendations of the DT, the physical is equipped with the capabilities to adjust itself in the physical world. This is done with various actuators. Sensors and actuators are the backbones of the DT (Tao et al. (2018)). Sensors sense the physical world, and actuators execute desirable adjustments. Sometimes augmented reality (AR) can be used as well to reflect some parts of the virtual product in the physical world. This can be used to view the real-time state of a product, as an example.

5. **Establish real-time, two-way, and secure connections between physical and virtual product**
   The enabling technologies here are network communication, cloud computing and network security. The DT is connected to the cloud through the network connection, then sends data to the "cloud" where the virtual product is. Then in the cloud, the virtual product can be developed, deployed and maintained in the "cloud", which means anyone with internet access can work with it. Lastly, the product data are directly and indirectly concerting user-product interactions, so secure connections between the physical product and virtual product is necessary.

6. **Collect all kinds of product-related data from different sources**
   According to Tao et al. (2018) there are four types of product-related data that should be processed by DT. These are product data, environmental data, consumer data and interactive data. Product data has consumer comments, viewing, and download records. Interactive data has user-product-environment interaction, such as stress, vibration, etc. IoT technologies can collect some of this type of interactive data in real time, and can then analyze it, and then make evaluations which are fed back into the first step. Thus, a loop to create a more functional virtual product is made.

Interestingly, Tao et al. (2018) mentions a new paradigm shift for bicycle use which is happening in China. There, they have share bicycles which are locked and unlocked through QR codes scanned with an application on a smart phone. Since the bicycles have location-based IoT technology, the user can open the app and find nearby bicycles for use. If they do need to use it, they can scan the QR code to unlock it, ride it where they need to, then lock it and pay for the ride through the application. According to Tao et al. (2018) this new "bicycle-sharing" (as it's called) has been a success case within public transportation, and shows how new IT technology, digital twins and IoT can help bring upon new changes in our daily lives.

2.3.2 IoT Architecture Patterns

With Intel's article about architecture patterns for IoT, it was an easy decision as to what architectural patterns fit the idea for our system. The patterns we will mostly focus on are Broker Pattern, and Publish and Subscribe. While it is a relatively new pattern, We will talk about the concept of Digital Twins in its own section.

Broker Pattern

As a baseline for this project, the architectural pattern we will be working with is the broker pattern. What this pattern does is define a run time component called a broker that acts as an intermediary between the client side components and the servers. As can be seen in figure 2.4 where a very simplified broker pattern diagram has been set up.

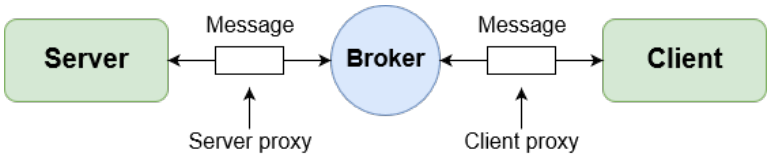


Figure 2.4: A simplified Broker Pattern diagram.

The problem this patterns solves is one where the service users don't necessarily know the nature and location of the service providers. This makes it easy to dynamically change the bindings between users and providers in real time.

However, this does not come without its downsides. With the use of a broker, there is an added complexity to the system where there might be a need for both client-side and server-side proxies to the brokers, and there might be a need to set up custom messaging protocols. The added complexity also makes it more difficult to test the system. With the added node, so to say, between the client and server, there's also an added latency in the communication and the communication is reliant on another node with being online. The broker is also the key point of attack in a security related manner, which means extra care needs to be put in to make sure the broker and the communication is secure.

While these weaknesses may seem daunting, the benefits of such a system outweighs the negatives depending on the use case of the system. To start off the is the added benefit of modifiability as to how the system interacts between the components. It's easy to change between the connections and add/remove links between the service providers and service users. All this without either party knowing about the change. This could be an added performance benefit where the broker can easily assign work to the least busy server. With this there's also the added benefit of availability in the case of a server going offline. The broker can then replace the failed server and the client continues its work without knowing.

There are many uses of this pattern, but we will mention of few of the bigger or more common ones. The fist widely used implementation of the pattern is the Common Object Request Broker Architecture CORBA (2018), CORBA for short, which is an open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. The more common uses of the pattern can be found in Enterprise Java Beans ORACLE (2018), EJB, and Microsoft's .NET platform Microsoft (2018), both of which are distributed service providers, and have a service-oriented architecture (SOA) approach.

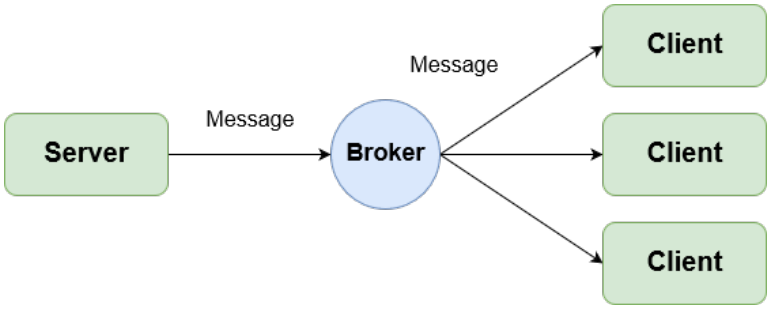


Figure 2.5: A simplified Multicast Pattern diagram.

Publish and Subscribe

Before going into the Publish and Subscribe pattern, we should briefly mention Multicasting. Multicasting is an extension of the broker pattern. How the Multicast pattern works is that instead of sending messages back and forth on a one to one basis, like a regular broker pattern, the sender sends a message to the intermediary, the broker, which in turn distributes the message to multiple recipients. This can be seen in figure 2.5, which is a simple version of the Multicast pattern. Not including proxies and other components relevant to the pattern.

A benefit from the Multicasting pattern is its bandwidth efficiency compared to one to one communication. The sender can simply send the message to the broker and not worry about using more bandwidth. As with the regular broker patter, the sender doesn't need to know who the recipients are. Intel does note that there are downside to the pattern, with security as an example, and the fact that the bandwidth save only matters if most or all values of the transmitted message is used.

Moving on, the Publish and Subscribe pattern is an extension of both Broker and Multicasting patterns. This is an event-based pattern and has an indirection layer between senders and receivers. There are independent processes or objects(namely also clients and servers) which react to events in the system, and thus reacts accordingly depending on the announcements. This is the pattern that most closely reflects the system being described in this report. A simplified diagram of this pattern can be seen in figure 2.6. Here the client has a subscription in the system and receives the event-based message sent by the server. The usage of this pattern in our system is further explained in chapter 4.

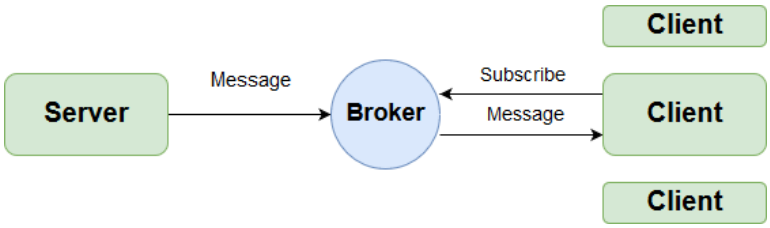


Figure 2.6: A simplified Publish & Subscribe diagram.

A benefit of the Publish and Subscribe pattern is the separation of sender and recipient where both sides are ignorant of the other side. This makes for easy change in the system in relation to change of the servers and consumers. This is however not without a cost. Given the indirection of the system, there's an added latency. In turn there is a run time performance reduction. This can affect the scalability of the system, depending on how it's implemented.

More often than not, in practice the pattern is realized by having a message-oriented middle-ware. The broker then takes this role. As the middle-ware, the broker manages the connections and information flow between the servers and receivers.

The pattern can take on a few different forms:

- **List-based publish-subscribe:** Here the publisher has a list of subscribers(receivers with an interest in the event). This makes for a less separated system with less run time overhead, but with the drawback of it being less modifiable.

- **Broadcast-based publish-subscribe:** Here the publishers have less to no knowledge of the subscribers. An event published and broadcasted. Subscribers then receive and examine if the event is within their interest before reacting. This can be inefficient if there are a lot of messages with no interest points.

- **Content-based publish-subscribe:** Here there's a distinction from the other two, which are topic-based variants. Topics are predefined messages and the receiver subscribes to all events within the topic. Content-events are associated with a set of attributes and delivered to a subscriber if those attributes match subscriber-defined patterns.

2.4 Open Source Software

Gaff and Ploussios (2012) defines Open Source Software (OSS) as a way to distribute a piece of software or application in the form of source code that other developer can view, modify, extend, and further distribute with or incorporated into their own software. There are, however, restrictions on OSS that anyone who decides to incorporate it into their own software need to be aware of. This is because it usually has a varying option of licenses it incorporates. All from more restrictive licenses to open licenses.

Buxmann et al. (2012) further defines OSS with the Open Source Initiative's (OSI) definition around OSS, which also sets some criteria that OSS needs to comply, as open source doesn't simply mean access to the source code. OSI gives nine criteria (which is also based on the licensing of the OSS):

1. **Free redistribution**
   The license linked with the OSS application shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several sources. There should be no royalty or fee for sales.

2. **Source code**
   The source code must be included and allow distribution of source code and compiled application. If no distribution of source code, the application should be well-documented with means of downloading the source code. Obfuscation of the source code is not allowed. Intermediate forms are not allowed.

3. **Derived software**
   The license must allow modifications and derived works, and must allow these to be distributed under the same license as the original software.

4. **Integrity of the author's source code**
   The license can restrict parts of the code from being distributed after modified, but only if the license allows for "patch files" with the source code for modification when the code is compiled. The license must explicitly permit distribution of software built from the modified source code. And the license may require the derived works to carry a different identifier from the original.

5. **No discrimination against persons or groups**
   Anyone should be able to contribute.

6. **No discrimination against fields of endeavor**
   Anyone should be able to use the program in any field of endeavor.

7. **Distribution of license**
   The license connected to the program carries over when there's a redistribution, without the need to add additional licenses.

8. **License must not be specific to a product**
   The rights attached to the program shouldn't depend on the program being part of a particular software distribution. If the program is extracted from said distribution and used or distributed within the terms of the program's license, then all parties receiving the program have the same rights as the original.

9. **License must not restrict other software**
   The license on one specific OSS must not place restrictions on other software that is distributed along that OSS.

Some of the more advantageous reasons to use OSS is that, since it's open, there are no royalties, or license fees. Usually, there's also a community of users that are continuously contributing with improvements, debugging, testing and updating the OSS application. This makes OSS good for quality and cost.

Muir (2005) tells a story in their journal about one of their first encounters with OSS as a concept. At an event they attended there was a hackfest they couldn't attend, where people were split into groups and were tasked with making an application. When at the end of the day they wanted a report on how the hackfest had gone, they were baffled by how quickly people were solving their issues and creating their applications. This was largely due to

these groups, those ahead of the others, had OSS solutions in the application creation. Which shows the power of using OSS.

But in the end, who owns the resulting product from the OSS? The original developers of the OSS or their work partners own the copyright in the OSS. Through licensing, these owners can make the OSS available for anyone to use under the terms they've specified. We mentioned licensing in OSI's definition or criteria of OSS. If someone wants to copy, modify, create a derivative, or distribute the OSS, then they either need to own the copyright, or do so under a license from the copyright holder. Failure to do so, even when the OSS is included in a larger package, means breach of copyright.

Without going too much into the specifics of the licenses used for OSS, According to Gaff and Ploussios (2012) OSS usually has three different categories of licensing. These are; restrictive(called "copyleft"), moderately restrictive(called "copycenter"), and permissive. While there can be a number of different licenses, they usually fall within these three categories. As can be guessed, these categories have different levels of what they restrict, with restrictive, as it implies, being the most restrictive license type. A lot of what these licenses restrict is defined by GNU's General Public Licenses (GPL).

OSS is with this generally considered a great resource and option, because if, say a company, wants to supplement and extend their existing software or hardware products they can use OSS. This will allow the company to focus on core proprietary functionality, while the OSS is developed by third-party (here communities or other companies).

2.5 Technology Acceptance Model

Technology Acceptance Model (TAM) explains how a new technology and the various aspects of it are received and used by the user. It says in its simplest form that the utility of the technology is the strongest influence on whether the technology is put to use, while how easy the technology is to use is less important. Davis (1989). We can use TAM to predict that an IT system experienced as useful by users will be used, even if they have to struggle to learn it. Conversely, a system that is easy to use and learn, isn't used if people doesn't see the benefits of it.

In the model, attitudes are assumed to predict intention of use, which can predict the actual use. Attitude toward use depends on two variables, perceived usefulness and perceived ease of use. Perceived usefulness says something about how much the user believes the technology will help improve efficiency, while perceived ease of use tells how comfortable the user is in using the technology. Both of these variables determine the user's attitudes to the system. The user's intention and perceived utility influence the individual user's use or discard of the system. Figure 2.7 shows the model that Davis presented in 1989. Since the announcement, the model has been introduced in several IT environments, but as many models it has received its criticism. Therefore, researchers have expanded the original TAM to TAM 2 and 3, to comply with the criticism as well as the ever-changing IT environment.

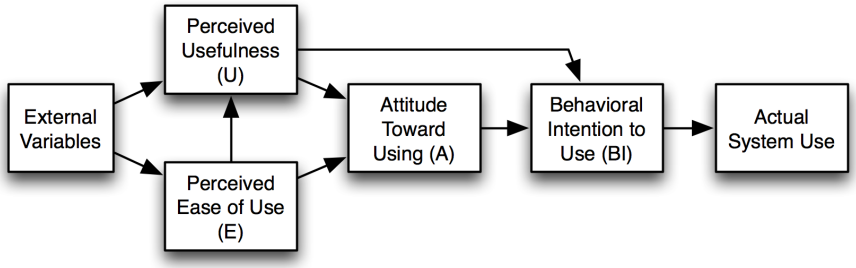


Figure 2.7: Technology Acceptance Model.

2.6 User-Centered Design

User-centered design is an iterative process that iterates over four phases, see ISO 9241-210 ISO (2010), figure 3.4. The first phase is about understanding and convey the usage. To understand the usage context, we use techniques such as, field studies, interviews, role-play or literature study. For the communication of the usage context, we use personas that are based on observations and interviews, as well as scenarios of a relevant situation. The second phase is about specifying the user requirements, here are also interviews and focus groups good techniques for specifying the requirements. In order to convey the requirements, a prioritized list of all the requirements with an explanation is a good option. The third phase deals with the actual development of the design solution. Here, several forms of prototyping are used, both low- and high-fidelity, as well as design guidelines. In order to convey the design solutions, it is common to do demonstrations of the prototype. Norman (2013). The final phase is evaluation of the design encouraged by the requirements given in phase two. Evaluation methods are, usability testing in lab, focus groups with feedback on solution, field tests and scoring forms such as SUS. See section 3.3.6 for more on SUS forms. The results of such evaluations can be presented in the form of a test report, summary report, or a data analysis.

Chapter 3

Research Methods and Design

This section will first address the theory and aspects of the research methods presented under chapter 1.6, with the figure 1.3. Then moving over to research design. In the research design section, we present all five research questions and methods we used to collect data for answering them. In this section, each research question is linked to the iterative processes of the human-centered design model, and to the research process model by Oates. The last part of the chapter, we will look at the validity of the research methods.

3.1 Research Methods

3.1.1 Qualitative and quantitative research

Qualitative research is about studying objects in their natural surroundings, and the desire to gain a better understanding. Quantitative research is concerned with finding a statistically significant relationship or for comparing two or more groups.

Qualitative data includes all data that is not a number, such as words, images, sound and likewise. These data are found in methods such as interviews, documents and observations. The desire here is to gain a deeper understanding of various phenomena such as human behavior. The data basis is usually large, as there is also the possibility of several explanations for a problem, rather than than only one correct explanation(Mcleod (2017)).

Quantitative data is data or evidence based on numbers. There are mainly data produced from surveys and controlled experiments. Analyzing quantitative data is based on finding a pattern in the amount of data. Tables and graphs are often used to visualize the data(Oates (2012)).

This study will use a combination of quantitative and qualitative research methods, but with a main emphasis on the qualitative.

3.1.2 Interviews

Interview is a special type of conversation between people. The conversations are usually not random as a person wants to collect information from several interview objects.

Interviews can be divided into three categories: Structured, semi-structured and unstructured. Structured interviews use predetermined questions. It is more like an oral reading from a questionnaire, where social interaction is used to clarify misunderstandings. In semi-structured interviews, there are prepared questions, but not all must be used and the order may vary. In addition, new questions that were not prepared can be asked during the conversation. In the case of unstructured interviews, the interviewer has much less control compared to the other methods. It is often started by introducing a topic, where the interview objects can freely talk about their opinions and what else they might think about the topic, with as few interruptions as possible from the interviewer. Semi-structured and unstructured interviews receive extra data that would not have appeared in a questionnaire, but are best suited for discoveries than checking.

The interviewer's role can have an effect on the data being collected. People may respond differently depending on who is conducting the interview. An interview may be easier if the interviewer and interviewees have the same background and a feeling that they understand each other. It is important to have good contact with the interviewees, listening and showing attention can be at the expense of writing down answers. Therefore, it might be wise to use video and audio recordings. Often, people can give the answers they assume the interviewer wants to hear. And this can be reinforced by the fact that people are often a little restrained when audio and video recording occurs(Walsham (2006)).

3.1.3 Prototyping

A prototype is a preliminary edition of a product. A realization of a design that stakeholders can see, touch and explore in the given environment. The limitation with the prototypes is that they often focus on a small characteristic of the product itself and disregard everything else. Prototypes can be anything from a small and advanced software code to being simple drawings and sketches. Prototypes are meant to be an aid to clarify communication between team members, and to explore design ideas with stakeholders and designers(Preece et al. (1994)).

One often distinguish on low- and high-fidelity prototypes. Low fidelity is far from the end product, but rather reproduces some aspects of it. The materials used for such prototypes are often cheap, simple and easy to change. For example, paper sketches and history boards. Low-fidelity prototypes are important in early development phases because they are simple, and encourage to explore and modify the possibilities that exist. Vague requirements can also be tested to see if they can be made concrete or removed. Another advantage is that the threshold for feedback on such simple prototypes is much lower than that of high-fidelity prototypes, and it is easier to get more constructive criticism from user tests.

The disadvantage of low fidelity prototyping is that they are not a part of the actual end-product, it is only meant for exploration. Therefore, it is important to evaluate them

against each other and see if the time and resources used, is considered worthy from such prototypes. It also requires a little imagination from users who are going to test such prototypes, as they often sit with a piece of paper in their hand and try to imagine it being a smart phone.

High-fidelity prototypes use materials that are likely to be found in the final product, and provide a clear picture of how the product will be. Such prototypes address some parts of the product, and are useful for detecting technical problems. These prototypes can also be reused in the end product itself, but they require more time and resources to complete. When testing high-fidelity prototypes, the threshold for feedback may be slightly higher than for low-fidelity prototypes. When users have an almost finished product in front of them, they can hold back the criticism because they think changing the prototype will take a lot of time and resources. Candidates can often get caught up in superficial things with the prototype than the content itself(Preece et al. (1994)).

It is common to look at prototypes that either cover a breadth of functionality or depth of functionality. Wide prototypes deal with many different functionalities, often low-fidelity prototypes, while depth prototypes detail in small areas, often high-fidelity. The type of prototype that is created depends on which original questions are the starting point for the prototyping.

Figure 3.1 shows a model to help with the choice of how prototypes must be designed, where each corner of the triangle represents important questions that are essential to the design of the system. Designers can use the model to separate design into three classes of questions that often require different approaches to prototyping. Implementation usually requires a functioning system to be built and deals with technical aspects of the system. Look and feel is about the user experience and requires a specific user experience to be simulated or actually created. Role is about what role the system should have for the users, and requires a context towards the objects to be created. Being explicit about which motives and questions must be answered makes the model an important aid in determining what kind of prototype to build. The model helps to visualize the focus of exploration(Houde and Hill (1997)).

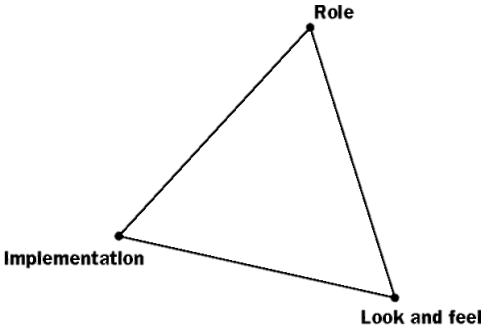


Figure 3.1: Prototype triangle

3.1.4 Usability testing

For testing how useful a system or product is, user testing is the most significant research strategy that can be utilized. In human-machine interaction design processes, this is a

major part, where the objective is to discover how helpful an item is to the intended user group, with the goal that they can increase the usability of the system. The term usability includes several concepts and the ISO standard: 9241-11:2018 defines usability as:

> *"Extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."*
>
> - ISO (2018a)

The definition highlights which characteristics must be measured to determine how useful a system is. The effect of the system refers to how precisely the user is able to complete the task on a system. Something that can be measured is for example: Number of tasks a user can complete, or how far a user came to a specific task. Efficiency is about how much work is needed to complete a task. This can be measured, for example, in the number of mouse clicks. The last characteristic, satisfaction, is about the user's individual attitude to using the system. This can be measured, for example, in an interview or with a questionnaire.

By constantly keeping users in mind, (user-centered design), it is possible to achieve these goals. Implementing a user-centered design process is about getting user feedback during the design and creation process. Always keeping in mind how the user solves this problem and consulting with users when in doubt.

When usability testing is to be carried out, goals and what must be achieved from the test must be set in advance. The tasks in the test must resemble the tasks that the developers should do in the final product, and the tasks must allow them to explore.

After the tasks for the test are made, participants must be recruited. Participants should be potential users of the system and should not be overqualified. By over-qualified developers we mean people who have had too much insight into the product at an earlier stage.

The number of participants being tested has a lot to say about how much error can be found using the tests. Figure 3.2 shows how many participants must be tested to find the desired amount of errors. Nielsen and Landauer (1993) has found that after testing 5 users they found 85% of the errors. Several claim that 5 is too small, especially if it is a large software program. The number of participants can always be increased to find more errors, but this will come at the expense of time and resources.

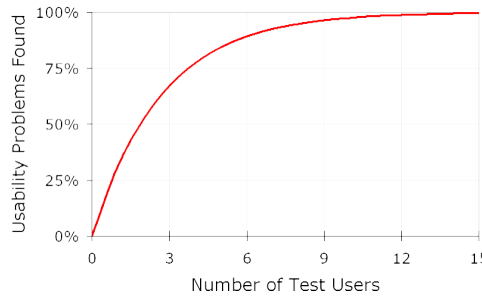


Figure 3.2: Errors detected given the number of user tests

The next point to be determined is the test environment. When a prototype is to be tested on users, natural surroundings can be a good solution, or an artificial environment can be set up in a lab. In order to get as good feedback as possible, the test environment should resemble the environment the system is intended to be used in. It may be wise to film the tests, but then the users must be informed and approve it in advance. The roles of a user test are important to determine. There are test participants, test leader and some observers. The test leader guides the participants throughout the test and acts as an interviewer to get the participants to come up with their thoughts about the experience of the product.

3.1.5 Guidelines for usability testing

When performing a usability test, there are several things that must be considered. A typical usability test includes several components. Tognazzini (1992) provides ten guidelines for developers and researchers on how a usability test should be organized:

1. Introduce yourself and any other persons who are involved in the usability test.
2. Explain the purpose of the test and specify that it is the product being tested, not the participant.
3. Inform the participant that he / she can quit the test at any time without any further explanation.
4. Describe the technical equipment set up for user testing. Explain the purpose of the equipment and how it will be used in testing.
5. Teach the participant how to think out loud to gain insight into the participants' thoughts
6. Explain that you will not be able to give the participant help during the test.
7. Describe the task given to the participant and introduce the product / prototype being tested.
8. Ask if the participant has any questions about the task before the actual test is starting.
9. End the test by letting the user comment on the product / prototype.
10. Use the results.

3.1.6 Questionnaires

Questionnaires is a widely used quantitative research method. A questionnaire is a predefined set of questions, assembled in a specific order. The data collected can be used for statistical analysis and is therefore widely used. Oates (2012). Questionnaires can either be self-administered or research-administered. When a questionnaire is self-administered, the participant completes the questionnaire without any communication with the researcher.

Self-administered questionnaires can either be given in paper or electronic forms. Research-administered questionnaires are very similar to structured interviews. The researcher is present and writes down the answers. The advantages of this method rather than self-administration questionnaires is that there can be more precise answers, especially if the participant does not fully understand the question. The disadvantage is that it is time and resources demanding.

System Usability Scale (SUS)

A System Usability Scale form, hereafter referred to as SUS. Is a good and quick way to provide quantitative measurements of usability after a usability test. The SUS form is a simple form consisting of ten statements, in which the participant should give an indication of how he or she agrees or disagree with statement. The participants response on a scale from 1 to 5, where 1 strongly disagrees and 5 strongly agree. See figure 3.3

Figure 3.3: SUS response

A SUS score is calculated based on how many points are given to each question. For the odd statements 1,3,5 etc. subtract one from the user response. For the even-numbered statements, subtract the user's responses from 5. This scales all values from 0 to 4, with four being the most positive response. To calculate the SUS score, all the answers for each user are summed and the sum is multiplied by 2.5. This converts the values to 0 to 100 instead of 0 to 40. A SUS score of more than 68 points is considered above average and slightly below 68 is below average. Sauro (2011). The SUS form is enclosed as appendix (D).

3.2 Validity of Research Methods

When considering all the data from the usability test, the group interviews and the questionnaires, it is important to look at the validity of this data. In this study, it is important to consider objectivity, transferability to other users, ecological validity and triangulation. Oates (2005).

Objectivity is about the extent to which the data collected comes from the interview subjects and participants in the tests, and not from the researcher itself. To prevent this, it is important to minimize unintended influence that the researcher may have over the participants and not ask leading questions.

It is also important that the data collected is transferable to other users. If there are too few users or not enough representative users in tests or interviews, the results will not be good enough for generalization. Oates (2005).

Ecological validity concerns whether the user tests are performed in environments similar to those in which the users will be using them. The closer to the natural environment the user tests can be kept in, the higher the ecological validity. The user tests shall be carried out in a usability lab, where the surroundings will be approximately the same as the real ones. This will provide high validity. Oates (2005).

Triangulation means that two or more methods are used to collect data in a study to check the results. If this is used, inconsistency in the data being collected can be minimized. For example, during a user test it may be observed, interviewed and given a questionnaire to the user to collect data. Oates (2005).

After the usability test, all data that is collected will be evaluated and discussed according to these criteria for validity in chapter 9.1.

3.3 Research Design

The research design defines how research methods were applied to the research questions in order to answer them. The ISO standard is not a research method as such, but a design framework. This has for us been useful for structuring our research process. The research questions are covering five stages of product development. First, the product requirements, followed by the technology stack and then the challenges with implementation towards the given technology. The last two questions focus on the usability of the product with the updated requirements found during the experiment.

3.3.1 Human-Centered Design: ISO 9241-210

ISO 9241-210, ISO (2010), Is an international standard that provides requirements and guidance for user- and human-centered design principles and activities throughout the life cycle of computer-based interactive systems. It is intended to be used by those managing design processes, and is concerned with ways in which both hardware and software components of interactive systems can enhance human–system interaction.

The ISO standard is not a research method as such, but a design framework. This has for us been useful for structuring our research process. The research questions are covering five stages of product development. See figure 3.4. First, the product requirements, followed by the technology stack and then the challenges with implementation towards the given technology. The last two questions focus on the usability of the product with the updated requirements found during the experiment.

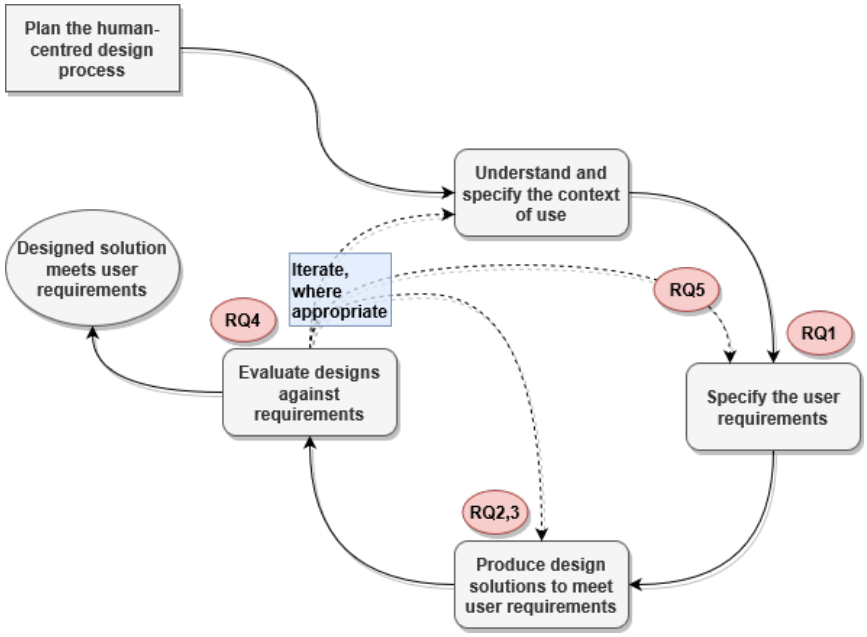


Figure 3.4: Human-Centered Design

3.3.2 RQ1: Requirements

What are the requirements for an open source architecture to support the development of pervasive exergames?

This research question falls under the activity "Specify the user requirements". The activity is about identifying the user's needs and functional requirements for different solutions. In human-centered design processes, user requirements should be set in relation to the intended context in use. The user requirements are a subset of the total requirement specification for an interactive system. In order to find the requirements for the architecture of an open source IoT platform, we first have to look at the criteria set by the EXACT project. Then we have to do literature review through previous master's theses associated with the project. In addition, we must do research on experiences in IoT architecture, where the focus lays in strengths, weaknesses, omissions or bias in the previous work. With enough information, we can compare and analyze the literature and previous experience. Our goal is to identify common elements, issues and technologies to find similarities, that will help clarify the requirements we search. This question falls under Oates *literature review* process. Figure 3.4 shows where RQ1 fits into the human-centered design processes.

3.3.3 RQ2: Technology stack

What existing software technologies are best suited to realize these requirements?

This research question falls under the activity "Produce design solution to meet user requirements". Design decisions will have a major impact on the user experience of a system. The design solutions should include an overview of the technology stack, user-system interaction and the user interface. The design solution should be made more concrete by, for example, creating scenarios, simulations or prototypes. To answer this question, the findings from question one, will help to come up with a design solution. In addition, the literature study will be expanded to cover various technologies and opportunities that might be best for realizing the requirements from question one. Part of the answer to the research question will also be based on experience gained by the implementation itself, and from the updated requirements from research question five. It is possible that technology found by literature search was not necessarily best suited to realize our requirements. Therefore, it is important to reflect on the technologies at both the implantation and after a usability test. This question fits the design part of the *design and creation* process under the literature of Oates.

3.3.4 RQ3: Implementation

What are the challenges of implementing such an architecture for a specific target technology?

This research question falls under the activity "Produce design solution to meet user requirements", and under the creation part of the *design and creation* process. The decisions made during implementation ties in with the design decisions made. The implementation solutions should tie the required software and hardware together in a sensible way, given the requirements of the system.

To answer this question, the findings in question one and two help in determining the best way to implement such an architecture. Research on various different technologies helps the realization of which technologies and softwares are the best fit for a system. All of these findings result in a developed framework and examples.

Many of the answers regarding this question will reveal itself in the implementation phase itself, including internal testing and reflection on what has been done wrong and what can be better, this from the development itself with the code-base and the use of the decided technologies. With the implementation phase itself spanning user testing and updating of requirements, as per questions four and five, some challenges also reveal themselves during these phases.

3.3.5 RQ4: Usability and Usefulness

How do potential users of the resulting framework assets its usefulness and usability?

This research question falls under the activity "Evaluate design against requirements". User-centered evaluation is an important and necessary part of human-centered system design. The evaluation is important for gathering new information on users requirements, and feedback on the design's strengths and weaknesses. Such an evaluation checks whether the user requirements for the design are met or not, and then compare several design solutions.

To answer the research question, two user tests will be held in the form of workshops. The user tests will be twofold, as it is interesting to look at two different work methods. One user group will work in groups of two, while the other user group will work one by one. Participants will be new to the product and have never tested it before. Each workshop will take about 3 hours including a 15 minute break in the middle.

During the user tests of the framework, the participants' thoughts, discussions and reactions are observed. This will provide the basis for qualitative data that can provide insight into what the participant feels about the framework. After testing the framework, participants will receive a System Usability Scale form (SUS) that they will answer individually. This provides the basis for quantitative data and provides a quick overview of how usable the framework was. Then, a semi-structured group interview will be held for collecting qualitative data. This will contribute to a more detailed understanding of the user experience. During the entire user test and interviews, video and audio recordings are made. This means that the test leader does not have to make notes through the interview, in addition we get the opportunity to analyze the user test afterwards. Figure 3.5 shows how the experiment will be used to generate data using interviews, observation and questionnaires, achieving good method triangulation. The procedure for the user test will be described in chapter 7, the agenda for the workshop is also described in chapter 7 and the System Usability Scale form is enclosed as appendix (D).

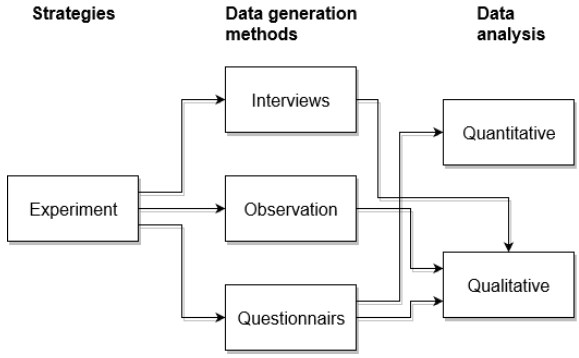


Figure 3.5: Model of research process for RQ4

3.3.6 RQ5: Updated Requirements

Based on the previous question; what are the requirements for such an architecture?

This research question iterates back to the activity "Specify the user requirements". In order to answer the question in the best possible way, we need to look at everything we have learned through the research process. Experience and motivation, literature review, design and creation and the results of the experiment. When looking at how new user requirements affect the existing requirements for architecture, we must reflect on more aspects than just the results from the usability test. This allows us to see how the results from the usability test affect the starting criteria stated by the EXACT project, and how new requirements can pose challenges in connection with implementation. By answering the question in this way, we get a more general view of the overall user-centered design process and we create a better basis for updating the requirements for architecture. Figure 3.6 is based on Briony J Oates's research process model, and it shows the entire overview of our chosen research process.

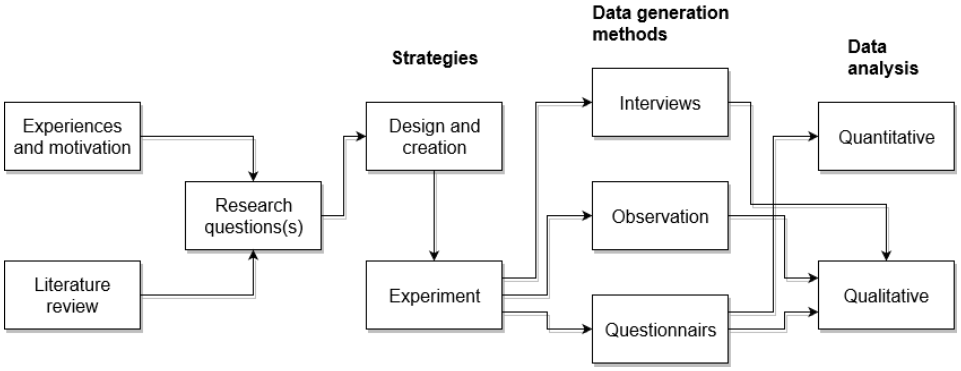


Figure 3.6: Model of the research process.

Chapter 4

Architectural Requirements

In this chapter, we consider the stakeholders for the framework, ASR's as a subset of the requirements. Followed by the main functionalists and functional requirements, the modified criteria of Johansens research, and in the end we look at quality attributes, scenarios and components and technical constraints.

4.1 Stakeholders

The term stakeholder is used to refer to any person or group who will be affected by the framework, directly or indirectly. Stakeholders include end-users who interact with the system and everyone else in a project who may be affected by its installation. Other stakeholders may be open source contributors who are developing or maintaining the framework. This definition is partly borrowed from Sommerville (2016)

4.1.1 Project team (Framework Developers)

The project team as stakeholder is the EXACT project team including us as master students. The project team is concerned about good cooperation and easily task dedication, as well as delivering an open source framework with high quality documentation. This is for the project to be easier transferred to new developers, both within the project and by any open source developers from communities such as GitHub.

4.1.2 End-Users (Game Developers)

End user is concerned about how easy and interesting the platform is. Game developers is mainly interested in the frameworks documentation, which provides information of the frameworks features and functionality, as well as examples. Further work developers is mainly interested in the overall architectural documentation including the updated requirements from usability testing.

4.2 Architecturally Significant Requirements (ASR's)

Architecturally significant requirements are the requirements that measurably affects a framework's architecture. This may involve both programming and hardware requirements. It's a subset of the architectural requirements, which affects the construction of a framework in quantifiable recognizable ways. Chen et al. (2013).

4.2.1 Main functionality

Physical game object with an instance

The most important driver for the functional requirements is the possibility of physical game objects being instanced as a digital instance. These objects can be viewed as a digital twin of the physical object. The state of the physical and the digital object should be synchronized. To meet this requirement we need to have a stable lightweight communication protocol.

Enable game engines to centralize the game logic

In order for a gaming engine to be able to control the central game logic in such a platform, it is important to adapt an adapter between the game engine and the communication layer. This adapter must transport the information between the gaming engine to the communication technology and then on to the physics objects.

Communication protocol

Communication protocol is a critical component of the system. It must be fast and easy at the same time as it must ensure that communication is understandable and that it appears as quickly as possible. We want to use Message Queuing Telemetry Transport (MQTT) as this is an ISO standard that meets the requirement for a publish-subscribe pattern we need.

DHCP router

To create a whole platform, we need a DHCP router that manages the wireless link between the different hardware we use. We want to use a Raspberry Pi with WiFi technology built-in. This will then act as a wireless router, with the ability to customize according to functional requirements. The operating system should be open source.

4.2.2 Criteria from Johansen

As a baseline we have taken the criteria from Johansen (2018) and modified them to fit our project. Where applicable, they've kept their numbering from the original criteria table. See table 4.1 for its current state.

Criteria	Description
C1.0 - Reduce implementation cost	The framework and the technology used should reduce the overall cost of prototyping pervasive exergames
C2.1 - IoT Flexibility	IoT technology should support different I/O modules to allow developers to customize the game objects to their game design.
C2.2 - Addressability	IoT technology should provide the ability to uniquely identify and address the device.
C2.3 - Device-to-Device communication	IoT technology should support the ability to transfer and receive data from different IoT devices.
C2.4 - Distributed vs Local use	IoT technology should support both local and distributed applications.
C2.5 - Scalability	The IoT technology should support the ability to be used in pervasive games with different amounts or sorts of devices.
C3.1 - Interoperability	The framework and the technology used should support communication between various IoT devices.
C3.2 - Connecting new devices	The framework should provide intuitive handling of connecting new devices both during development and run time.
C3.3 - IoT Flexibility	The framework should allow developers to create custom interaction and visualization of different virtual I/O components that mirror traditional physical I/O modules.
C3.4 - Game logic centralized	The framework should support running the game logic outside of the IoT devices.

Table 4.1: Modified criteria based on findings from Johansen (2018)

4.2.3 Functional Requirements

ID	Requirement	Priority
FR1	Working MQTT Broker on a RPi	High
FR1.1	DHCP Server on RPi	High
FR2	Create individual Arduino configurations	Low
FR3	Connect class instances with physical objects	High
FR4	Automatic setup out of the box	Medium
FR5	Synchronization between physical and virtual objects	High

Table 4.2: Functional Requirements

4.2.4 System Quality Attributes

We have classified the most relevant system quality attributes for the project into primary and secondary according to their relevance. These quality attributes must be reflected in the IoT Platform architecture.

Modifiability

This is one of the main quality attributes of our platform. This should be reected in the architecture. It should be easy to add or remove new features in the future. For example, replace the game engine for one that fits better for the game to be developed.

Interoperability

Real-time synchronization between game objects is the main feature of the platform, and interoperability is therefore an important quality attribute, so that both physical and virtual game objects maintain the same state at all times. Additionally the framework and technology should support communication between various IoT devices without knowing the specications of each device.

Usability

Usability is an important quality attribute for this platform. The platform will be used by game developers with different levels of programming skills. It is important that the platform clearly documents functionality, so that those who use it easily understand functionality and also makes it easier to further develop the platform.

Performance

A weak spot in all small hardware is performance. It is therefore important to develop a platform that uses efficient technology, which doesn't drain too much resources from the hardware. This arguments for why performance is another signicant quality attribute for

this IoT platform. It will significantly impacts the device's user experience. Our platform will seek to deliver as low latency as possible.

4.2.5 Quality Attribute Scenario

Quality attribute scenario is a schematic overview of the specific quality attributes, we use them as a means of characterizing the quality attributes. These forms consist of six parts; source of stimulus, stimulus, artifact, environment, response and response measure. Figure 4.1 shows a graphical illustration of a scenario. For our architecture we have chosen modiability and interoperability as the primary quality attribute, performance and usability as secondary quality attributes. Following sub classes address the quality attributes individually through scenarios presented in tables.

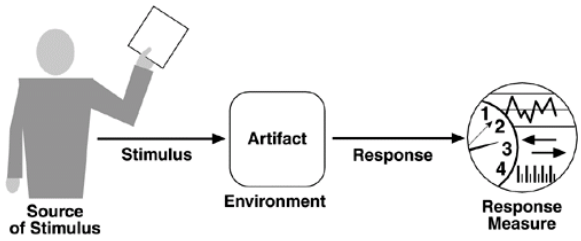


Figure 4.1: Quality attribute scenario Bass et al. (2003)

Modifiability

ID:	M1
Source:	Developer
Stimulus:	Wishes to change the game engine
Artifacts:	Code
Environment:	Design time
Response:	Makes changes
Response Measure:	Unknown

Table 4.3: Modifiability 1

ID:	M2
Source:	Developer
Stimulus:	Wishes to change to a different MQTT broker
Artifacts:	Hardware/Software
Environment:	Design time
Response:	Make changes
Response Measure:	Within less than eight hours

Table 4.4: Modifiability 2

Interoperability

ID:	I1
Source:	Developer
Stimulus:	Request to setup a new devices
Artifacts:	Code and Arduino device
Environment:	Normal operation and design time
Response:	Unity application handles request
Response Measure:	All request are handled in the correct format and order

Table 4.5: Interoperability 1

Performance

ID:	P1
Source:	Unity application
Stimulus:	Receives data from client
Artifacts:	Arduino device
Environment:	Normal operation
Response:	Data is used to update virtual objects
Response Measure:	Expected 40 messages per second

Table 4.6: Performance 1

ID:	P2
Source:	Unity application
Stimulus:	Send data to client
Artifacts:	Arduino devices
Environment:	Normal operation
Response:	Data is used to update and operate physical objects
Response Measure:	Expected 40 messages per second

Table 4.7: Performance 2

Usability

ID:	Usability 1
Source:	User/Game developer
Stimulus:	Implement the framework
Artifacts:	Code
Environment:	Design time
Response:	Framework works according to the documentation
Response Measure:	At least four hours

Table 4.8: Usability 1

4.2.6 COTS - Components and Technical Constraints

Raspberry Pi and Raspbian Linux OS

Raspberry Pi (RPi) a small cheap single-board computer, with enough resources to run the features the platform requires. The requirement for an open source platform puts constraints on which operating system can be used. In our project we will use the Linux distribution Raspbian.

ESP8266 running Arduino

ESP8266 is a small Arduino chip with wireless communication technology Wi-Fi and Bluetooth. It is designed to run the Arduino platform, a platform based on open source. It makes it easy to use with sensors and actuators. This is cheap technology that makes it possible to customize any instance of a game object. Its constraints are the Arduino platform, and that the development environment comes with a C / C ++ library called "Wiring".

Eclipse Mosquitto - MQTT broker

Mosquitto MQTT Broker is a lightweight message protocol using the pattern publish-and-subscribe. It makes it easy for low power devices to communicate between different IoT

devices. The Mosquitto project utilizes a C-library for implementation of MQTT clients, which adds constraints to the types of chipsets we can use. Fortunately, this fits well with the ESP8266 and the Arduino platform, but will require an adapter plugin towards the game engine.

Chapter 5

Technology

5.1 Technology Stack

Recall from chapter 2 where we covered the IoT technology stack at the end in section 2.3. In this technology stack we had three main elements that made up the stack. Those elements were listed in figure 2.3, but as a reminder, these three are:

- IoT Cloud
- Connectivity
- Things/Devices

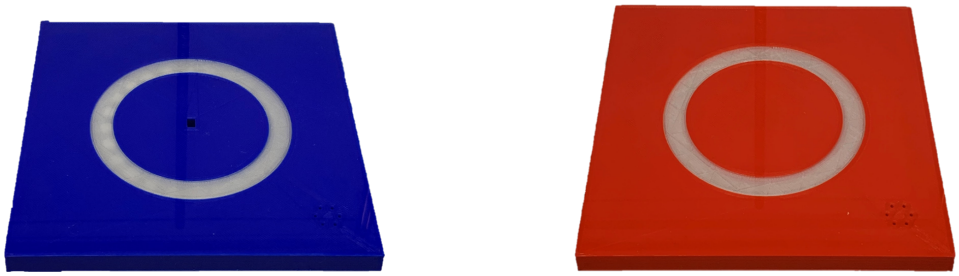
There are three more elements that span the layers of the stack. However, these elements are outside the focus of this project as we're not integrating either enterprise software, nor third-party information. An argument could be made for the inclusion of the security element, but that too is outside of our focus.

5.2 Hardware

For the hardware side of things, aside from using breadboards to connect various components for testing, we had two main hardware setups made specifically for the platform and testing.

Tiles

These two hardware configurations are what we dubbed in the code RedTile and BlueTile. They are essentially plates with various hardware inside, and can be seen in figure 5.1. The two tiles have slightly different configurations.



(a) Blue Tile **(b)** Red Tile

Figure 5.1: 3D printed prototype tiles, blue and red.

The blue tile seen in figure 5.1a has a ring of LEDs(Ring Light), a tone player, a distance measurer(Time of Flight), and a gyro and tap sensor(IMU). While the IMU has the functionality for a gyro, it is not enabled in this hardware configuration.

The red tile seen in figure 5.1b has almost the same configuration as the blue tile, with a ring of LEDs, a tone player and a gyro and tap sensor. What differs in this configuration is that it lacks the distance measurer, but has the gyro enabled.

ESP Arduino

Powering both tiles is an ESP8266 Adafruit Huzzah Feather from Adafruit (2018), a type of Arduino micro-controller. The ESP8266 is a low-power and light chip that was specifically made to be small and light, and thus perfect for portable micro-controller usage, according to Adafruit. This chip was selected due to its low price tag, and because of it's shape and size makes it a good fit for the components being developed for the prototype. The chip also supports built-in WiFi and Bluetooth modules. This is perfect as you only need the chip itself and not other separate parts as you would with other Arduinos, as can be seen with Sparkfun's board list (Sparkfun (2018)) of different Arduino boards fit for different projects.

Raspberry Pi

For the prototype we are developing, and the wish for a portable system, there was a need for something that could be used with the system as a connector to the Arduinos. While a laptop or even a second Arduino could be used for this purpose, a small computer like the Raspberry Pi (Foundation (2018a)) seemed like a good fit. In our case, the Raspberry Pi 3 Model B. As the Raspberry Pi itself is a small computer, it serves well as a connector for the Arduinos and beyond that. It runs the Raspbian Foundation (2018b) operating system, which is based on Debian, a Unix-based operating system. This is a small and light operating system, which is needed given the hardware of a Raspberry Pi.

5.3 Software

5.3.1 Connectivity

In this section we go over the software used in the connectivity layer in the development process of the prototype. These will also be used in the final prototype of the framework.

MQTT

MQTT ISO (2016), Message Queuing Telemetry Transport, is an ISO (2018b)-compliant Client/Server messaging protocol that solves the problem of the Publish and Subscribe architectural pattern described in chapter 2. It is lightweight, open and simple, and designed to be easy to implement, which makes it ideal to use for communication in constrained environments like the IoT context considering the small code footprint or network bandwidth it requires.

It requires an ordered, loss less, and bi-directional network protocol like TCP/IP to work and has the following base features:

- One-to-many message distribution and decoupling from the Publish and Subscribe pattern.
- A messaging transport that is agnostic to the content of the payload.
- Three qualities of service for message transport:
  - *At most once:* Messages transported the best it can within the constraints of the operating system. This means message loss can occur
  - *At least once:* Messages can be assured to arrive, but duplicates can happen.
  - *Exactly once:* Messages can be assured to only arrive once.

5.3.2 Application

Unity Game Engine

Unity is a game engine made by Unity technologies that supports both development of 3D and 2D games. What a game engine is, Unity Technologies puts nice and shortly:

> *A game engine is the software that provides game creators with the necessary set of features to build games quickly and efficiently.*
>
> - Unity (2018)

The game engine supports exporting for up to 27 different platforms, including consoles, mobile, and traditional computers. Games developed in Unity are developed using C#, but over its lifetime Unity has had support for the UnityScript system Boo, a Unity-exclusive

scripting language, and JavaScript as well. In 2014 Boo-scripts were deprecated and removed, while JavaScript has been in use up until Unity 2017 where its deprecation process started.

Unity is used by small indie studios and professional AAA studios alike, and is considered one of the, if not the, biggest game engines at the moment. This is due to how intuitive it is to use and its cross-platform capabilities. It also has good support for importing assets from other programs like Blender, 3ds Max, Photoshop, and more. However, while Unity does support basic shape objects, any complex modeling has to be done in third party programs like the ones mentioned earlier. It also sports the Assets Store which is an assets library where developer can share their custom assets to help the community.

Given the use of C#, Unity also has the ability to import other standard .NET libraries, making it easy to implement protocols and libraries such as MQTT.

Some Other Game Engines

One of the biggest game engines is Unreal Engine (EpicGames (2018b)). This is a game engine created by the game studio Epic Games (EpicGames (2018a)), the studio behind arguably the biggest game of 2018(as of writing this report). It supports development in the programming language C++ and their custom UScript language, and also their custom Blueprints which allows non-programmers to use their engine to create.

Another big game engine is Cryengine (Crytek (2018)) from the studio Crytek. This is one of the more powerful game engines in terms of pushing "realistic" graphics and visuals. It supports the programming languages C++, C# and the script language Lua.

We also have a relatively new game engine called Godot Godot (2018) from the Godot studio. This is a free to use and open source game engine for 2D and 3D game creation. This game engine supports visual programming akin to Unreal Engine's Blueprints, but also its own scripting language Godot Script and the programming language C#.

5.4 Proposed Technology Stack

While the original three-layer technology stack proposed by Wortmann and Flüchter (2015) works fine for our project, there are different ways to setup a technology stack, and we have decided then to modify this technology stack a little. This modified technology stack can be seen in figure 5.2.

The main difference between our suggested technology stack base and the original proposed technology stack, is adding a layer for the object abstraction which covers the connections types between devices, like WiFi, Bluetooth, or similar. We have also changed the name of the IoT Cloud layer

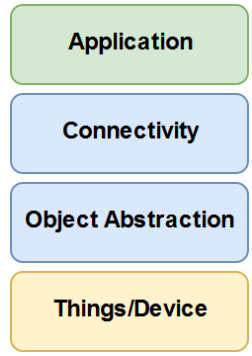


Figure 5.2: A modified version of the technology stack seen in chapter 2.2.

to Application to better fit the use of a program made with the Game Engine Unity. Though the name of the layer is changed, the use is practically the same as the proposed technology stack describes it; a management software.

With this modified technology stack organization, this gives us the following technology stack, which can be seen in figure 5.3.

Briefly, this gives us the following organization; Unity program as a data and device management software, the brain of the system. MQTT as the connection handler between devices in the system, where most of the messages go back and forth between devices and Unity. WiFi as the connection type, and how devices are connected together. Lastly, Arduinos and a RPi being the main IoT device objects with actuators and sensors. Their implementation will be covered in the next chapter.

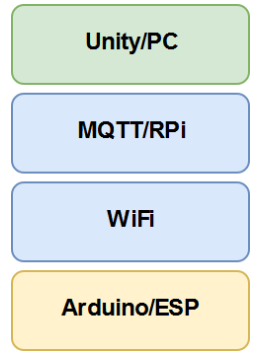


Figure 5.3: Proposed technology stack for this project.

Chapter 6

Implementation

The aim of this chapter is to give an overview of the architecture being developed, with how the components are set up, their roles in the architecture, and their communication. As well as covering some examples of the use of the framework developed in Unity, and our challenges during development.

6.1 Implementation Overview

The figure 6.1 presents a general overview of the architecture being developed. As can be seen, the architecture is modelled after the Publish and Subscribe and the Digital Twin patterns described in chapter 2. With the illustration of how the system is set up with MQTT, it can easily be expanded with other components that uses MQTT as a messaging protocol. Pattern-wise we see a raspberry Pi as the messaging broker in the system with different clients connected to it. The digital twins connected to the PC/Unity components are digital representations of the physical Arduino components. How these are set up will be described below. All the communication is done over either WiFi or Ethernet, as required by the MQTT messaging protocol.

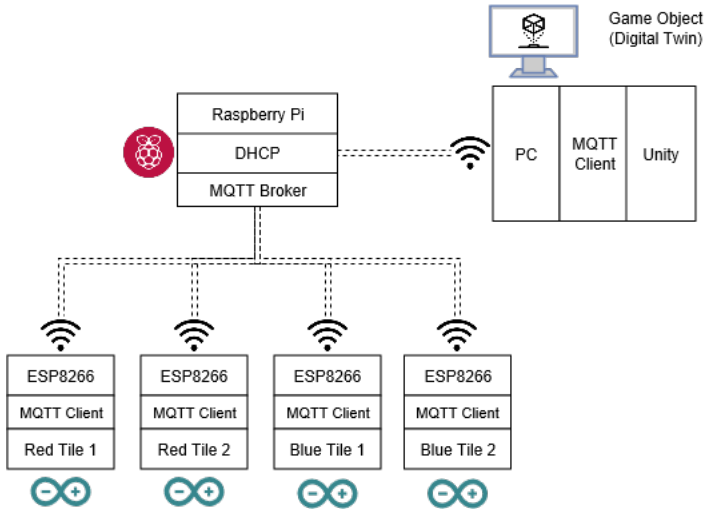


Figure 6.1: Architecture Overview

6.1.1 Component Roles

Raspberry Pi

The Raspberry Pi is the broker in the pattern and is set up with the Mosquitto message broker which implements the MQTT protocol. Ecclipse (2018). This is a very lightweight message broker, which makes it perfect to use in low power single board computers like the raspberry pi, and is widely used in IoT projects. In the system, all messages passes through the Raspberry Pi since it works as a broker, it's powerful enough to run a DHCP server as well, turning it into a router and a connection point for all the other devices.

PC/Unity

The PC component is a computer running the game program compiled from Unity. While Unity can compile to Linux, which makes the Raspberry Pi a potential target for running the game itself if it's light enough, a distinction has been made to properly separate Unity as its own component.

This component in the architecture runs the game logic and will function as a server or service provider for the rest of the system, as per the Publish and Subscribe pattern. With the game, the logic of the system, and digital twin data, all run here. While all the MQTT messages run through the broker, the other components will not talk to each other and will act separately with all handling of the messages being done in Unity.

Arduino

The Arduino component(denoted by the Arduino logo in figure 6.1) is a reactionary component in the system. Aside from going online and connecting to the MQTT broker and

subscribing to certain messages, this component mostly awaits messages from the broker to update its actuators (lights, sound, etc), or sends a message over MQTT if a user interacts with one of the input sensors. As the specific Arduinos used in this system does not have Ethernet connections, they run on WiFi to connect to the DHCP server on the Raspberry Pi. The configurations and how the Arduinos are set up can vary from component to component, but their role remains that of a client, as per the Publish and Subscribe pattern.

Game Object (Digital Twin)

The Game Object component is a sub-component of the PC/Unity component. These are Unity-specific Game Objects which have class objects that hold information based on the various Arduino components. This can be information on the whole configuration of the device it's a Digital Twin of, or the states of the various components, such as button and light states, or methods for controlling the different physical actuators connected to the Arduino. This acts as a synchronized digital mimic of the physical device.

6.2 Component Connection

While the Publish and Subscribe pattern is used, the broker in the system is only there to relay the messages between the various components. This is only there to help realize our system and architecture, it is not necessarily a point of interest connection-wise in the architecture. There are then two connections to focus on; the connections between Arduino and Unity, and Unity and the Digital Twins.

6.2.1 Arduino and Unity

The connection between an Arduino and Unity has most of the information about the type of message sent set in the topics of the MQTT messages. As MQTT and the Publish and Subscribe pattern works with the client subscribing to certain messages/topics and the service sending out messages fitting these, Unity will send messages fit for each Arduino with different types of messages specified in the message topic.

Within MQTT, the topics are flexible and a client can subscribe to all or some sub-topics within a topic. Let's take an example of this with the topic a client subscribes to being "device". MQTT allows you to have different layers of topics separated by a delimiter "/", and also subscribe to all topics within a topic with the symbol "#". Say a client wants to subscribe to all topics within the "device" topic, the client would have to subscribe to "device/". On the flip side, a client can subscribe to specific messages like "device/update".

Message Overview

Previously in the section we talked about how the "#" character in the MQTT message topic helps the device subscribing to all messages within a topic. In this architecture this is used for both Unity and the devices. Unity subscribes to all the sub-topics within the "unity/#" topic. The Arduinos do the same, with their MAC-addresses being their individual identifier

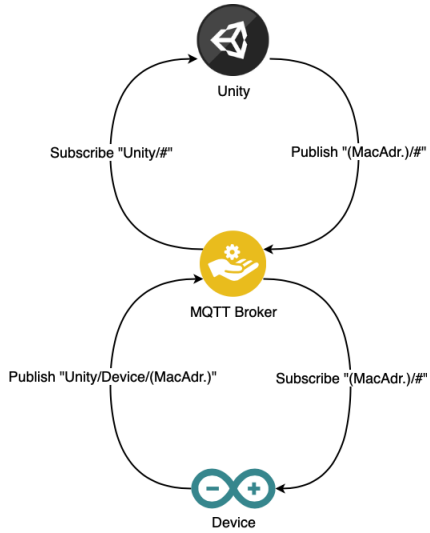


Figure 6.2: Subscribe and Publish overview

in the topic. The Arduinos all send their messages within the "unity" topic, while Unity sends its messages within the topics of the unique identifiers from the Arduinos. This message structure can be seen in figure 6.2.

In this system, there has been set up five different types of sub-topic messages handled throughout. These are as following:

- **Event message:** A message sent from an Arduino when a sensor has an event happening.
- **Action message:** A message sent from Unity to make a change in an actuator connected to an Arduino.
- **Get message:** A message sent from Unity when it needs to get the value from either an actuator or sensor. May have limited use as sensors send their values as they update.
- **Value message:** A message sent from an Arduino with the value of a connected component when it receives a get message from Unity.
- **Ping message:** A message couple between Unity and an Arduino to check if they're still connected to each other. Will be initiated by Unity at certain intervals if needed. Not a message type the game developer needs to worry about as it's controlled by the system on its own.

The general message flow can be seen in figure 6.3. The ping messages are not included here as they are simply periodic messages happening within the system and not handled based on what the user of the framework programs.

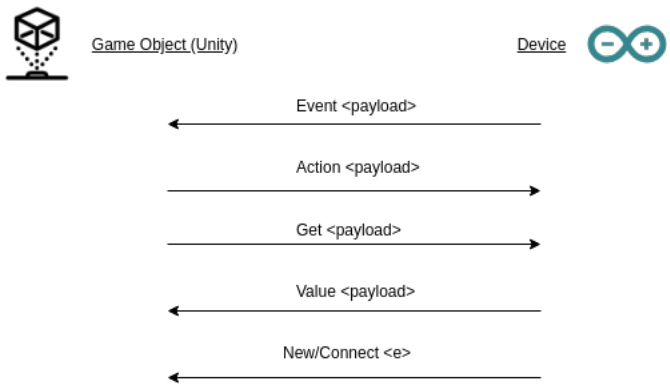


Figure 6.3: Information flow overview

Connect Message

The connect message is a message type sent from the Arduino when it connects to the MQTT network. The connect message's structure includes the unique identifier of the Arduino in a sub-topic. During this connection message, the configuration ID of the Arduino is sent as well. This specific ID sets which digital twin the Arduino is linked up with in Unity. The configuration of the Digital Twin and Arduino is decided based on how the game developer creates their own configurations, or if they use one of the few pre-made configurations included in the framework. The connect message is formed as: *unity/connect/MAC-Adress/configurationID*. However, before the connection message is sent, the Arduino needs to subscribe to its specific unique identifier, making it ready to receive Unity's response to the connection message. Due to the limitations of our system in its current state, Unity is required to be online before the Arduino goes online. The Arduino's connection set up can be seen in the following code snippets.

The Arduino retrieving its MAC-Address and subscribing to it:

```
clientIDstr = WiFi.macAddress();
client.subscribe((clientIDstr + "/#").c_str());
```

With the subscription set up, the Arduino is ready to publish its connect message with the structure from earlier, with the MAC-Address and configuration ID. A couple of the various configurations will be covered later. The publish message being sent can be seen in the following snippet:

```
client.publish(("unity/connect/"+clientIDstr+"/"+configID).c_str(), "1");
```

Given that Unity is already online prior to the Arduino sending its message, means it has already subscribed to its topic of choice. when Unity goes online, it simply subscribes to the "unity/" group of topics, making it ready to receive messages from all the other components. This is done in the following snippet:

```
client.Subscribe(new string[] {"unity/#"}, new byte[] { MqttMsgBase.
    QOS_LEVEL_EXACTLY_ONCE });
```

The client here is the MQTT Client object inside Unity from the MQTT library used in the framework.

When Unity receives a message, an MQTT Handler class handles the MQTT message, extracting both the MAC-Address and the configuration ID before linking up a Digital Twin object for the specific Arduino within Unity. When the Digital Twin setup is done, it is ready to send and receive other message types. These messages are mostly dictated by unity.

The twin setup on the Arduino's connect message is done with the following code snippet, where the message is first split, then handled:

```
void handleMQTTMessage(object sender, MqttMsgPublishEventArgs e) {
    String topic = e.Topic;
    String[] topicSplit = topic.Split('/');
    if(topicSplit[0] == "unity") {
        if (topicSplit[1] == "connect") {
            deviceConnect(topicSplit);
        }
    }
}

private void deviceConnect(string[] topicSplit) {
    bool linkListFull = true;
    List<GameObject> objectList = gameLogic.getTwinObjectList();
    foreach (GameObject obj in objectList) {
        if (!obj.GetComponent<TwinObject>().getLinkStatus()) {
            obj.GetComponent<TwinObject>().linkDevice(topicSplit[2]);
            linkListFull = false;
            break;
        }
    }
    if (linkListFull) {
        sendDeviceMessage(topicSplit[2] + "/ping", 0);
    }
}
```

The MQTT handler class takes the message, extracts the Arduino's MAC-Address and the configuration ID, checks the Digital Twin objects created to see if they have the appropriate configuration setup the Arduino can be linked to. If it does manage to find a Digital Twin for the physical device, it responds to the connect message with a ping message.

Ping Message

The ping message is a message that is sent periodically from Unity to a device to see if said device is still connected to the broker. While the MQTT broker does have a ping message of its own built into Mosqitto(the MQTT software running on the Raspberry Pi), some level of control is needed on the Unity-side of the system. This can be useful when a device disconnects as that Digital Twin will be freed up to give room for a different device to take its place if the previous one doesn't come online again.

On the Unity side, the device is sent a ping message at a certain interval to make sure it's still alive and connected. If using the the Unity code structure, the classes that are connected to the game objects in unity have their own update function called every frame, which makes ping timing relatively easy, putting the ping message method call there with

some simple time management. This also makes the time interval easily changeable by the game developer working on the system. This ping timing is set for all Digital Twin game objects and a call to a ping message method is made where the twin object calls on its reference to the MQTT Handler to send the ping to the linked device.

```
public void sendPingMessage(string deviceID) {
    mqttHandler.sendDeviceMessage(deviceID + "/ping", 1);
}
```

On the Arduino side of things, it's a simple matter of taking the received ping message, extract the message type to call the ping message method and send the required response back to Unity. Usually the ping message is sent with a payload of byte "1", but if the Arduino connects to the system and Unity doesn't have any available spots for that Arduino configuration, a payload of "0" will be sent instead to let the Arduino know it hasn't connected and can, as an example, go offline again. Both the connect message's response ping message and a regular ping messages are the same and are handled by the same ping message method on the Arduino side, seen in the follow code snippet:

```
void ping_event(byte* payload){
  Serial.print("Ping event received!");
  if ((char)payload[0] == '1') {
    Serial.print("Pinging back!");
    client.publish(("unity/device/" + clientIDstr + "/ping").c_str(), "1");
  }else{
    WiFi.disconnect(true);
  }
}
```

Action Message

The action message is a message sent from Unity whenever it wants to trigger some sort of output in the Arduino. This could be everything from an LED, buzzer, or any other physical component the Arduino is configured with that Unity has an implementation for.

The message structure is built from whatever component Unity wants to change. In similar fashion to the ping message's structure of *deviceID/ping* the action message, the action message is structure as *deviceID/action/component/component-output*. The *component-output* part is what's being changed on the component. Eg. on an RGB LED that could be the state or the color of the light. The action message method can be seen in the follow code snippet:

```
public void sendActionMessage(string componentName, byte[] payload) {
    if (mqttHandler != null) {
        List<byte> temp = new List<byte>();
        temp.Add((byte)payload.Length);
        temp.AddRange(payload);
        addActionMessageToBuffer(componentName, temp.ToArray());
    }
}
```

There's a check to see if the MQTT Handler object reference exists so the Unity programs can be tested without connecting to the network if necessary. The action message buffer method call is there to build the action message buffer which is used to combine all

action message called within one frame in Unity into one single action message to send to the Arduino.

For the Arduino, it's a simple matter of filtering the message to the action message method call and from there, extracting which component and what to change on the component. As the Arduino code base(c++) doesn't have a simple way to split strings, this is done with a loop and some if-else statements. The following code snippet shows a short example of updating an LED's state.

```
void action_event(char *topicElement, byte *payload) {
  while (topicElement != NULL) {
    if (strcmp(topicElement, "led") == 0) {
      topicElement = strtok(NULL, "/");
      if (strcmp(topicElement, "state") == 0) {
        if (payload[0] == 1) {
          lightsOn();
        } else if(payload[0] == 0) {
          lightsOff();
        }
      }
    }
    topicElement = strtok(NULL,"/");
  }
}
```

Event Message

The event message is a message type sent from the Arduino whenever an input or sensor is triggered. As an example, this could mean a button press or release, a distance measure, or any other form of input. MQTT supports sending messages (Kathrin (2015)), which means any type of sensors with a continuous change in the input could send a stream of messages, making the change in Unity fluid.

A quick example is looking at the code for a button input on an Arduino. Here there button is a c++ class with an update function which reads the physical state of the button. Then the button's state is checked and a message is sent depending on the change in the button state. Similarly, a continuous input would send a message if a change is found.

```
button.update();
if(button.pressed()){
    client.publish(("unity/device/" + clientIDstr + "/event/button").c_str(), "1");
}else if(button.released()){
    client.publish(("unity/device/" + clientIDstr + "/event/button").c_str(), "0");
}
```

For Unity, the incoming event message is taken in and filtered by the update method to the correct device's twin component object. The components values are updated based on this event message.

The first method for this functionality is the *deviceEvent* method which is called from the initial method call from the message being received. In this *deviceEvent* the incoming message is further filter to the specific Digital Twin object it's meant for. This twin object is called from the list of all twin objects with an on-going connection in the MQTT network.

```
private void deviceEvent(string[] topicSplit, byte[] payload) {
        TwinObject to = getObjectByID(topicSplit[2]);
        if (to != null) {
            to.eventMessage(topicSplit, payload);
        }
}
```

```
public virtual void eventMessage(string[] topic, byte[] payload) {
    pingTime = 60*60;
    EventMessage msg = new EventMessage(topic[4], topic[5], payload);
    updateComponent(msg);
    onEvent(msg);
}
```

Here the *eventMessage* method on the Digital Twin object class is called, which takes the message, creates an EventMessage class object that prepares an event object and passes said object on to the *updateComponent* method.

In the *updateComponent* method, the components are individually updated based on the values in the EventMessage object. This is a methods that's overridden by the configuration classes that inherit the Twin Object class. This means the user of the frameworks needs to updated the device components on the digital side themselves, but it also gives more general flexibility. The *updatedComponent* code is listed further down.

Get Message

The get message is very similar to a standard get message call in any polling based programming. In this case, the get message is Unity asking the Arduino for the value of a specific component. The Arduino responds with a message containing the value for the specified component.

As with the action message function call, the get message function call is similar, with a few parameters and message building before being sent by the MQTT handler to the Arduino.

```
public void sendGetMessage(string componentName)
{
    if(mqttHandler != null){
        getMsgBuffer.Add(new MessagePair(componentName, new byte[]{0}));
    }
}
```

As with the action message, every time the get message builder is called, the message in question is added to the get message buffer to handle multiple messages on the same frame in Unity, giving the Arduino time to handle messages.

Value Message

The value message is the Arduino's response to the get message from Unity. This is essentially the returned value from a get method call in any polling based programming. The value of the physical component attached to the Arduino(or other types of messages depending on the configuration) is retrieved and returned to the calling object. In this case,

the value is sent back to Unity in the form of a "value" return message over MQTT and handled as a regular update call on the Unity side.

```
void get_event(char *topicElement) {
  while (topicElement != NULL) {
    if (strcmp(topicElement, "imu") == 0) {
      topicElement = strtok(NULL, "/");
      if (strcmp(topicElement, "rotation") == 0) {
        client.publish(("unity/device/" + clientIDstr + "/value/imu/rotation").c_str
    (), rotation);
      }
    }
    topicElement = strtok(NULL, "/");
  }
}
```

As can be seen in the code snippet above, the get message is handled similarly to an action message, but a value message(similar to an event message) is sent immediately. As can be seen in this example, the value of a measured rotation is returned. In cases where there's a constant change in values and where the values are measured continuously without break, a get call is better than a continuous stream of messages. This depends on the sensor type as some only return measured values at certain times, some return measured values all the time, and others send off a single signal.

On the Unity side, the value message is received and handled the same way as an event message, calling the *updateComponent* method.

6.2.2 Unity and Digital Twin

While this isn't an external connection between devices like the connectivity between Unity and an Arduino, this is an internal connection between the data object of the Digital twin and the rest of the Unity code. Without going into detail listing every function of the Digital Twin objects, we can look at the more important ones.

We've briefly mentioned it, but the *updateComponent* method is the main way to update the components on Digital Twin objects. By taking parameters for the EventMessage class object built from the message received over MQTT, this method does the work of sorting out which components needs change, and then call the components' functions to make the appropriate changes to the component. Depending on how the component is implemented, and what type of component it is, the component will do a method call on Unity to make and send a message to the Arduino connected with the Digital Twin object. In some cases, an update on a component will also fire a Unity message method call to the game object, meaning users can have event based update methods like "OnTapped", or similar, where updates can happen without the user needing to poll for values from sensor components. The following code snipped shows an example of an *updateComponent* method.

```
protected override void updateComponent(EventMessage e) {
    if (e.component == "timeofflight") {
        timeOfFlight.setDistance(e.value);
        timeOfFlight.setMeasuring(e.state);
    } else if (e.component == "ringlight") {
        if (e.name == "state") {
```

```
7               ringLight.setState(e.state);
8           } else if (e.name == "color"){
9               Color tempColor = new Color(e.payload[0] / 256.0f, e.payload[1] / 256.0f,
      e.payload[2] / 256.0f);
10              ringLight.setColor(tempColor);
11          } else if (e.name == "numOfLeds"){
12              ringLight.setNumOfLeds(e.value);
13          }
14      } else if (e.component == "imu") {
15          if (e.name == "tapped") {
16              imu.setTapped();
17          }
18      }
19  }
```

The other major method in the connection between the Digital Twin objects and Unity is the *linkDevice* method. It was shown in the part where the new connection type message was explored. This is the method that is called when a device is connected to the network. This device is linked up with a digital representation of itself, namely the Digital Twin object. The *linkDevice* method itself is very short, and triggers a ping message to the physical device letting it know it has connected properly, as can be seen in the following code snippet.

```
1  public void linkDevice(string deviceID) {
2          this.deviceID = deviceID;
3          this.linked = true;
4          sendPingMessage();
5  }
```

Lastly, we have the method calls in the Digital Twin object that build messages that are sent from Unity to the Arduinos. All the Digital Twin objects have a reference to the MQTT Handler class, so the object just calls on the the main send message function with the message built inside the object. Different method calls has been implemented for ease of use, though the message structure is simple enough where only one or two methods could have been enough for the functionality. The following methods has been implemented, with various versions with different parameters, but the following listing of the methods has the standard parameter usage:

- *sendPingMessage*. Parameters: device ID.
- *sendActionMessage*. Parameters: device ID, component, payload.
- *sendGetMessage*. Parameters: device ID, component.

6.3 Device Configurations

The plan for the framework is to have various preset device configurations people can use with certain Arduino setups, and also the possibility to easily make custom configurations. During the development of the framework we've explored a few different configurations for testing purposes.

Cube 1: LED

Cube 1 is the first configuration we made and is noted as "cube1" in the Arduino code. This is a simple Arduino setup with a single LED connected to it. The LED has simple controls where it can be toggled, set to blinking, or changed blinking speeds. The blinking functionality is only available when the LED itself has been set to the "ON" state.

Cube 2: Button and Potentiometer

Cube 2 is the second configuration we made and is noted as "cube2" in the Arduino code. This is a sensor-focused Arduino setup with a button and a potentiometer. The button sends event messages on press and release, while the potentiometer has a stream of messages if it has a change over a certain value. A variable holds the current potentiometer value and could be retrieved with a get message with a few tweaks.

Tiles: Red and Blue

The tiles have been explored in a previous chapter and are the last configurations we tested during development. They're noted as "redtile" and "bluetile" in the respective Arduino codes. The both have a good mix of sensors and actuators connected, which makes for a good exploration and testing tool for the framework, as they both have continuous update components and single event components. The actuators connected, ring light and tone player, are also good to test various degrees of messages from Unity.

Other Configurations

The users of the framework can also set up their own configurations by extending the Twin Object class. The base class makes it easy to integrate other configurations into the system, with the logic of the custom configurations all being up to the users of the framework. The components are added the same way as a normal Unity component is added to a Unity game object, with *AddComponent* method calls. An example of a custom configuration setup, without the *updateComponent* method, can be seen in the following code snippet. This is a custom configuration with an LED and a button. In this snipped the meat of the *update* and *updateComponent* methods have been removed, as well as getters for the components.

```
1  public class Cube3 : TwinObject {
2    private Led led;
3    private Button button;
4
5    public override void Start () {
6      base.Start();
7      configName = "cube3";
8      button = new Button(this);
9      led = new Led(this);
10   }
11   public override void Update () {}
12   protected override void updateComponent(EventMessage e){}
13 }
```

6.4 Examples

In this section, the different examples created during the implementation process will be described.

6.4.1 Controller and Output

The controller and output example is the first and simplest test done during the implementation. This example consisted of two configurations for the devices in the system, namely a controller and output. Both configurations were set up on breadboards to quickly and easily test the physical side of the system.

The output configuration is the simplest of the two and consist only of a single LED(cube1). For this particular example the LED has three modes of operation; off, on and blinking. Off and on are fairly self-explanatory. Blinking, on the other hand, is more a sub mode of the 'on' mode, as to have the LED blinking, it needs to be set to the 'on' mode. The blinking itself has four different speeds that are controlled by the controller device. The blinking speeds are denoted by the numbers zero through four, and is handled by Unity.

The controller configuration consists of a button and a potentiometer(cube2). On a change on either the button or the potentiometer, the device sends a message over MQTT to Unity which handles the operations and updates the output device. The button controls the LED on the output to toggle between the modes 'off' and 'on'. The potentiometer sets the blinking speed of the LED where the lowest value is not blinking.

6.4.2 Sample Tile

The Sample Tile example is based on something the supervisor created running locally on a single tile device(see chap. 5 for the device). While not all of the functionality was ported over, enough was done to have a functional test of proper continuous data streams both from the physical device and Unity.

For this example the Ring Light LEDs light up when the IMU sensor registers a tap on the device and a distance measured from the Time of Flight sensor. For the IMU tap detection, a signal is sent to Unity which turns on the LEDs for a split second before turning them off again. This action takes first priority as you could still tap the device while measuring a distance. Which leads us to the distance measure. Here, a value is sent continuously from the device to Unity with the distance measured. Unity takes this distance measured and sets how many LEDs on the Ring Light should be used and turns them on accordingly. It also sets the color of the LEDs depending on the distance measured.

This example was especially helpful in showing potential users the various run time delays of the system. Both to show the limitations of the system and also what's available to them.

In figure 6.4 we can see the flow of events from the tap signal on the physical device to the Ring Light's reaction to the tap event. From this, it can be seen that Unity does all the handling of events and updates the physical device with the values set on the digital device. Similar flows of events could be drawn from the events of rotating the tile to light

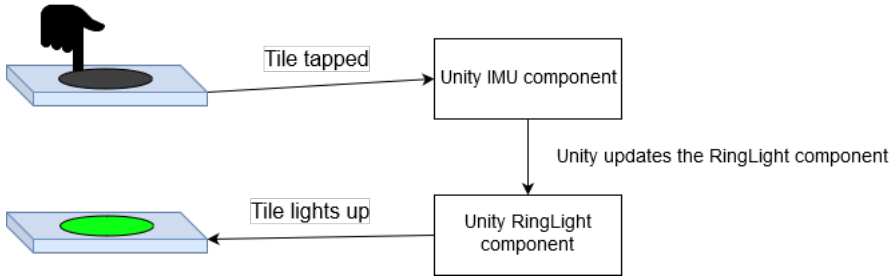


Figure 6.4: Sample Tile flow of events when tapped

up various single LEDs on the Ring Light depending on the rotational direction, or with the distance measured on the time of flight sensor giving different number of LEDs lit up with varying colors. Not included in this figure is the event of resetting the Ring Light to a default state after any action.

This example was implemented with one single class for the tile device(blue tile) used for the example, as no other classes were necessary to handle the events and information updates on the tile. However, since there were priorities in the output of the device, we needed some control, so a SampleTile class which inherited the blue tile's class was created. In this class all the necessary MQTT handling was added, as well as some polling on the various components of the device to make sure the priorities were intact. This can be seen in the following code snippet.

```
1            if (imu.justTapped()) {
2                ringLight.setColor(Color.green);
3                ringLight.setNumOfLeds(ringLight.getMaxNumLeds());
4                ringLight.setState(true);
5            } else if (timeOfFlight.getMeasuring()) {
6                ringLight.setColor(Color.blue);
7                int calcLeds = (int)((timeOfFlight.getDistance() / 200f) * ringLight.
    getMaxNumLeds());
8                ringLight.setNumOfLeds(calcLeds);
9                if (ringLight.getState() == false) {
10                   ringLight.setState(true);
11               }
12           }
```

There is also some handling to turn off the light on the tile again when nothing happens, which has been excluded from the code snippet above.

6.4.3 Follow the Light

The last example created during testing was the Follow the Light(previously called Follow the Red Dot) game concept that Johansen also had in his proof of concept, see figure 2.2. This example was also used as the base for a task the test users had to create during the usability testing detailed in chapter 7. For simplicity, only the tap events from the IMU, the tone player and the Ring Light's light, color and number of LEDs functionality are used in this example.

This example game is very simple with, after all the devices connect to the system, one of the devices lighting up and playing a sound to start off the game. When said device detects a tap from the player, a signal is sent to Unity which selects one of the other devices(Unity shouldn't be able to select the same device twice in a row) and sets that device to light up and play a sound.

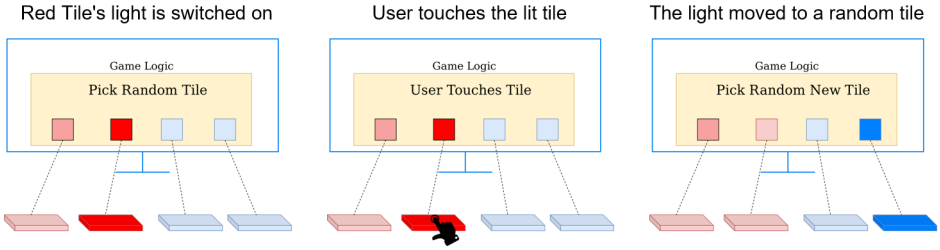


Figure 6.5: Flow of Follow the Light example

Figure 6.5 shows the flow of the Follow the Light game as described in the previous paragraph in its simple form and the example pre-prepared for testing of the framework. However, there is a more complex version of this that was developed as well. With the selection of the tiles and user input events being the same, there was also an added difficulty in the form of a countdown shown on the tile with a decreasing number of LEDs being lit up. When said countdown reaches zero, the game is over and all the tiles light up blinking in a different color for a while before restarting the game.

For the development of this example, two red tiles and two blue tiles were used to signify the usage of different devices, even though the two of them are quite similar. With this setup we developed the game two ways; one with everything in the game logic class using the tile class configurations as they were, and the other with the game logic class as setup for the game and initiating the game by selecting the first tile with the tiles extending from the base tile configurations to contain some logic.

6.5 Challenges and Reflection

While the examples explained above are relatively simple, the development of them wasn't without problems, having issues with both the hardware and the software. Most of the issues we had, aside from figuring out exactly how to code things and set things up, were on the Arduino part of the framework.

Setting up MQTT on all devices and getting them to communicate was never any issue, but on the Arduino side we had some issues with how the libraries themselves were set up. Seeing as we didn't want to tinker with the libraries too much, and the publish method call for the MQTT client required a c-type string, we had to go through some small hoops to convert various variable types to the required variable type. This was mostly converting to raw data as that's what was required for the MQTT message. we could have done some editing on the libraries themselves for the Arduino, but decided against it to keep whatever

coding we did limited to our own code.

While our work is focused on the software, we did have some issues with the hardware during testing. These issues arose mostly during the time when we connected the same components used in the tile configuration on breadboards before the tiles themselves were completed. We thought both during and after testing with the breadboards that the boards themselves were the issue, as there might have been some bad connections between the wires, boards and components. The IMU and Time of Flight components used a certain amount of power and had to be connected to the 5V output of the Arduino. Due to either the wiring or the breadboards themselves, we often got issues with those components where the Arduinos wouldn't detect them. Though, after the tile setups were completed with the parts soldered together, this issue was gone.

Lastly, more is written about this in the last chapter in the section about further work, but we could have handled the code differently for Unity. As the code is right now, it's mostly a polling based framework with few ties to how Unity usually does things with their game components steer away from how Unity developers want their system, as we'll get a closer look at in the next chapter about our testing of the framework.

Chapter 7

Usability Testing

7.1 Research design

The usability testing as an experiment is a strategic research method we use to obtain the data we need to answer research questions four. There will be two user tests, as it is interesting to look at two different working methods. The first user group will work in pairs, while the second workshop the users will work individually. During the user tests of the framework, the participants' thoughts, discussions and reactions are observed. This will provide the basis for qualitative data that can provide insight into what the participant feels about the framework. After testing the framework, participants will receive a System Usability Scale form (SUS) that they will answer individually. This provides the basis for quantitative data and provides a quick overview of how usable the framework was.

7.2 Planning

This section details the recruitment process of the participants, the location and equipment used during the tests, a description of the test prepared for the participants, and details about a semi-structured interview.

7.2.1 Recruiting participants

For this study of the framework being developed, it's important to get participants for testing within the target group, namely software/game developers. If the number of participants within the target group isn't satisfactory, students can fill the missing spots given they have experience with game development or Unity, or have had a specific game design subject at NTNU.

Access to developers withing the target group was limited and wholly dependant on the participants' interest in the project and whether they have time, as we can't expect people to take time off from work.

Through the school/supervisor's connections, our supervisor got in contact with Work Work, a workplace of several smaller companies where some of them are game developers. With this we got a time slot during their lunch where we could present our project and recruit participants.

The students were recruited through an email sent out to an email group consisting of the students who had a game design course at the university during fall 2018. This part of the recruitment process was done as there was not a satisfactory number of participants from the Work Work recruitment phase. Aside from the requirement of having had the subject, there was also a requirement that the students were studying Informatics or Computer Science and had some experience with Unity. The students filling these requirements are also close to actual software developers and game developers reducing the loss of validity from the testing considerably.

For supervising the process of the workshops with us, we had our supervisor help with introductions and controlling the workshop. And our co-supervisor helped with recording the workshops for use in later review.

With this we had two workshops with a total of 3 participants from the Work Work recruitment phase and 3 participants from the student group, and our supervisors helping out. One of the students was moved to the first group so we could run two pairs of two and two, and singles the second workshop.

7.2.2 Location and equipment

For the workshops we were able to use the UX-lab, Svanæs (2015), at the university as our location for the testing. The lab has the ability to record both sound and video of the testing area, letting us observe our testers without being in the room watching over them.

As we were only testing a software framework, we didn't need much of equipment. For the workshops we needed two sets of raspberry pis, tiles and computers. As the tiles were made for our testing during development as well, we could supply the workshops with both raspberry pis and tiles. The computers, monitor, keyboard and mouse included, were borrowed from the university. Everything else in terms of tables and seating, etc, was either in the UX-lab already or borrowed from the university.

The UX-lab has two main rooms, seen in figure 7.1. One for observation, and one for testing. In the observation room there is equipment for controlling the recording equipment, and monitors to watch what is recorded. In the testing area are 4 cameras with microphones for the recording, the area we used to set up our own equipment, and where most of the workshop took place.

We had some refreshments - coffee, water and some snacks - placed on a table by one of the doors in the testing area.

By the refreshments, to the left in figure 7.1, we had some chairs to be used by the participants during various parts of the workshop, facing a monitor. To the middle of the room we made an equipment setup with the two groups on each side of a separator wall.

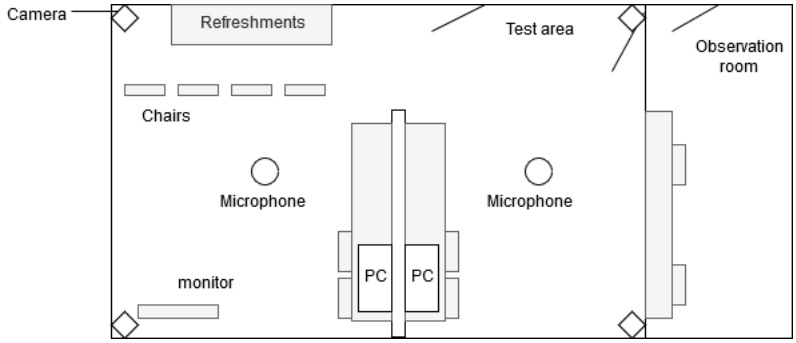


Figure 7.1: Layout of the UX-lab with computers, tables and separator walls set up.

7.2.3 User tests and tasks

For these workshop, we were testing the usability of the framework we created. The participants had access to a wiki we have created, and the rest of our code to look at. Due to the nature of what we're testing, it works a bit different than regular usability testing where you can easily see how the users handle what they're working with.

For most of the time during the workshops, the participants were on their own. This was to test how easy it was for said participants to get familiar with the framework supplied, and how quickly they could get to a point where they could properly both explore and work with the framework.

After the participants had some time to familiarize themselves with the framework, they were given a task. This task was a continuation of the example in chapter 6.4.3, here dubbed Keep the Light Alive. To make sure the task was solvable, we did the task ourselves before the workshop. The point of the task wasn't necessarily to finish creating what we wanted them to, but to get some form of results and see how they worked with the framework. We will have a closer look at how various participants worked with the task in sections 7.4.2 and 7.5.2.

For the task Keep the Light Alive, the participants were supposed to make the light count down, showing how much time the player has left with the number of lights on the Ring Light. As can be seen in figure 7.2, the game follows a few general steps. Some of the steps are very similar to the regular Follow the Light example detailed in the previous chapter, but with a few added steps. New additions to this updated game are the steps 2, 3a and 4a. For step 2, the countdown on the selected tile starts as soon as it's selected. If this countdown reaches 0, as seen in step 3a, the game over phase is initiated which should make all the tiles blink a few times for a set period in step 4a before restarting the game back at step 1.

7.2.4 Semi-structured interview

At the end of each workshop, we had semi-structured interviews. It is important to get the participants' opinions and experiences with the framework, as this can help guide further

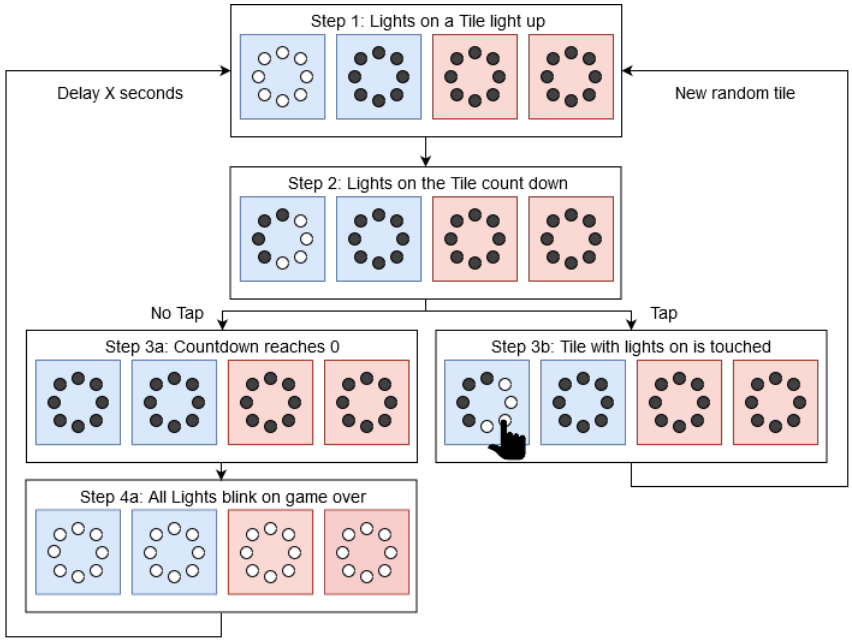


Figure 7.2: Keep the Light Alive gameplay steps

development of the framework before an eventual finished product. As the interview sessions are semi-structured, they allow for changes in order of questions and follow-up questions from the interviewer. For these interviews we were mainly interested in usefulness, usability and context of use, with questions within these categories. As all participants, and supervisors, of the workshop were Norwegian, the interviews and questions are all in Norwegian. The following questions have been translated from their original language. The questions were based on the TAM model Davis (1989) and ISO9241-11(ISO (2018a)) on usability.

- **Usefulness:**
  - Do you think you could use this framework in a project? Explain.
- **Ease-of-use:**
  - Was the framework easy to use?
    - What was easy? Examples.
    - What was difficult? Examples.
    - What could have been done differently? Examples.
- **Fit with context of use:**
  - How does the framework fit with how you work when you design games? Explain.

- **Summary:**
  - The best part about the framework.
  - The worst part about the framework.
  - Would you recommend this framework to your colleagues?
  - *If no:* What would need to change for you to recommend it?

7.3 Procedure

This section will detail our schedule for the two workshops we had, see table 7.1. Everything in both the workshops were scheduled to take about 3 hours, from 4 pm to 7 pm. As mentioned in the section about the user tests, before the workshops were held we did the task given ourselves to make sure parts, if not all, of it were possible to do within the given time frame. The second workshop was as similar to the first workshop as we could manage.

Time	Activity
16.00 - 16.30	Introduction - Purpose of the workshop - Architecture - Demo - Declaration of consent
16.30 - 17.00	Preview of code on GitHub
17.00 - 17.15	Task introduction
17.15 - 17.30	Break (Questionnaire)
17.30 - 18.30	Task: Keep the red dot alive
18.30 - 19.00	Group reflection
19.00 - 19.10	Hand out of gift cards

Table 7.1: Schedule of the workshops.

For the introduction of each workshop, we first introduced the project and the purpose of the workshop. Mostly what we were making and what we wanted to test. This lead into a brief talk about how the architecture of our framework works. A demo was then done to show the framework, and something made with the framework, in action. For this we used the last two examples detailed in chapter 6, the Follow the Light and Sample Tile examples. Underway we were open for questions from the participants in case something was unclear during the explanations. Lastly, the introduction was rounded off with the participants signing the declarations of consent.

After introductions, it was time for the participants to get familiar with the framework. For this we had the workstations each group would use. The first workshop split the participants in groups of two and two, while the second workshop split them individually, one and one. For the review process of the framework, they had access to the GitHub where the code was stored, and also the accompanied wiki pages also located on GitHub.

We held the task introduction before the break to give the participants some time to think about, discuss and plan their work with the task. For this, we held a small presentation

of the task before letting them have their break. As we needed to find a good time for the participants to answer a questionnaire about their use of frameworks, what platforms they've developed for, etc, this was done during the break as well. For the break, we also reminded the participants about the refreshments in the room. Consisting of some biscuits, fruits, coffee and tea, and water.

After the break, it was time for the user test with working on the task given. For this part of the workshop, all the supervisors left the test area room to observe the participants from the observation room. The participants were told that they would only get help with technical difficulties, e.g. with the hardware.

After the time was up for working with the task, we handed out SUS forms for them to fill out, before starting the semi-structured interviews. The interviews were recorded as it would be useful to go back and review the feedback from the participants. For this interview, we were mainly interested in what was both good and bad about the framework, and if anything needed to change, both in general and if they were to use or recommend it.

7.4 Results workshop 1

In this section, the results from the first workshop, where we had four participants split into groups of two, are detailed. We will look at some of our observations during the workshop, the code from the participants and how they decided to solve the task, some information we got from the group interviews, and the results of the SUS.

7.4.1 Observations: Problems and challenges

Given the hands-off approach to the task solving part of the workshop, the participants had to solve any software issues of their implementations on their own.

For the first workshop, there were no real issues we had to solve for the participants other than once. In this instance, one of the groups had an issue where they used the sample code supplied by us where an IP-address was set. This led to the group getting null-pointers and not being able to run the program at all. Since this wasn't a very obvious error for them, this led to some confusion about their own code and of how they used our framework. After fixing the error for them, the other group was notified as well about the IP change.

The above error isn't an error that's necessarily very clear to the participants when developing using the framework. On the layout that can be seen in figure 7.1, the raspberry pis acting as brokers in the systems are placed next to the respective PC setups. Both supplied with their own screens. However, when not directly working with it, and the task for the participants was to work on the Unity code alone, it becomes very unclear how to solve such a problem unless they were told about it beforehand.

Interestingly, during the framework- and code-review part of the workshop, one of the groups dove straight into the code and navigated their way through, finding out which methods were called from where, and the structure of the framework. This group consisted of both of the game developers. The other group took their time reading the wiki-pages we

had prepared, and spent considerably less time looking at the code itself. It was around this time it became clear that there was a considerable thought-process difference between the two groups, which led to two very different approaches to the task solving following this, and the game developer group probably developed a deeper understanding of the framework itself just from navigating all the code.

7.4.2 Code analysis

In the previous sub-section, we mentioned how the two groups had different approaches towards solving the task given, where they were to take the concept of the Follow the Light game in the examples and make the Keep the Light Alive game we introduced to them before their break in the workshop. We had told them they could either start over, or expand upon what we had already made. Giving them the possibility to copy and paste code if they needed, almost encouraging it.

While there might be an experience and framework knowledge gap between the groups, as discussed in the previous sub-section, the group getting closest to solving the task, creating what we wanted them to create, was the group with the students. This is probably due to how each of them are used to working, and their though processes on what they wanted to do.

The first group took what we said to heart and expanded one of the existing examples that came with the framework. This gave them an existing base that worked out the gate, and they only had to add the missing features the task required them to create. While they didn't finish, they had a working countdown on the tile objects, and got furthest in completing the task.

The other group with the game developers had a different approach to the whole task. Initially they had a sine wave going on the system to test the change in number of lights on a tile. They told us this was to get a better understanding of the framework and the system at hand, and to get more familiar with the delays in the system. Continuing this, they didn't necessarily go directly towards solving the specific task with adding the missing features to one of the existing examples. They were thinking bigger and started implementing their own game handler, with needed classes. This had an obvious effect on how far they got with the task, but it was valuable input in seeing how different people worked towards the same task. With their game handler they could have potentially used it for other games, while the other group went for a more specialized task solution.

7.4.3 Group interviews

While the SUS scores we'll present in the next part gives a good indication of how usable a system is, it's also important to get concrete feedback from the users on what was good and bad about the framework. Semi-structured interviews aids in this. Instead of having all the participants separately, they were all interviewed together so they could speak up if comfortable and add to what others said if they felt they needed to. We will abstain from quoting what the participants said as it wouldn't necessarily be accurate with a translation.

Usefulness

For the usefulness aspect on its own, we were interested in whether they could see themselves using the framework either in their jobs or some form of projects.

For the students, they couldn't really comment on the aspect of using the framework in a working situation, but they both expressed interest in Internet of Things, and potentially using the framework on a private basis. One of them noted how they struggle with setting up Arduinos and such, and connecting them. They could see themselves using this framework, given that it helps with such things.

The two developers both saw they could hypothetically use the framework if they had an idea where they could use it. One of them noted how the framework is, in a way, split in two. On one hand you have the things that you need to set up, and then on the other hand you need to use the things with the framework. We do see how this could be a potential entry barrier for some. They also noted how they sometimes work with clients, and it's useful to have something like this in their toolbox if someone comes in and wants to gameify something, and they spar back and forth on development ideas.

The two developers also referenced some examples where you could have games developed with this framework. A simple and obvious example is a whack-a-mole game. The Follow the Light game is very similar to this. Another example is to gameify exercises in football where the players run back and forth between boxes, which is a typical exercise in football.

Ease-of-use

For the usability questions, we were interested in the general aspect of whether the framework was easy to use. What was easy, difficult and what needed changes. Something of note is that when we went to Work Work where the game developers work to hold our presentation, we handed over the link to our code repository to those interested as a hook to garner even more interest. The two developers did note that, in terms of validity of the testing, they did have some prior knowledge of the framework going in as they did look at the code in the repository.

All of them had various different ways of saying it, but there was a lack of commenting and documentation in the code itself, making the framework a little difficult to get into. For some, it was the distinction between the game objects themselves, from the example code we gave them, and the core device within the framework. They struggled to see the difference in the game object and the class object being a twin of the physical device. They also wanted to see information about what kind of physical components a device had in the documentation. E.g. how many lights and other components. In general, they all wanted more documentation both in the code, and about the devices themselves, which is something we obviously should have had ready before the workshop.

Something of note as well, is that one of them wanted a higher abstraction level where they didn't have to worry about specifically the components themselves or other things, and just have an abstract class they could refer to and just say "I want the lights to be on". But this is also something some of the others did themselves with adding interfaces to their

classes to make their classes more abstract and have common methods they could call on.

While there were issues with the documentation and abstraction levels of the code, they all had some sort of consensus that the framework was simple and because of that (relatively) easy to understand. The game developers noted how the framework was functional out of the box and that things they didn't need to see, like network handling etc, was hidden from them and they didn't have to worry about it. They could just start developing without too many worries. One of the game developers said they liked the component system we had implemented in Unity. From this, they all agreed they could see how flexible the framework could be, even if it's very simple in nature.

Fit with context of use

For this part the we wanted to know how the framework would fit into how they as developers work when designing games.

While one of the developers liked the component system we had, they also came with a change they wanted when we asked about how well the framework fits into the work flow of a Unity developer, and this brought up the component system Unity already has. They said that if they could make a request then and there, it would be to have the system mimic the Unity component system in terms of adding components to game objects, as we now have everything in code and more like general C#.

Summary

For this part of the interview we wanted to know some of the best and worst parts of the experience with working with the framework, or just about the framework itself, and whether they could see themselves recommending the framework to a colleague and any changes needed for that to happen.

For the best of the framework, they all had a general consensus about the fact that the framework was relatively easy to understand and start using, and also the fact that it was a very simple framework. The two students noted that they could see there was a lot of freedom with the framework if they had time to explore more. The game developers were happy that the framework was what it was, and not something more complicated. Often when they start using new SDKs(software development kits) and plugins, there's often a lot to learn, and they were happy this framework was pretty simple and lightweight. They did note that they would've liked to see how it would've been to work freely during the workshop instead of working on the given task to see what could be done with the framework.

For the worst of the framework, they all agreed that there needed to be a better level of documentation for the framework, as it did take some time to read and understand what each method did. One of the student wanted something different from the examples of a finished game in terms of tutorials. Something more akin to instructables where you have a small code snippet focusing on an individual functionality. There was also an expressed want for a more Unity-like event system when variables change instead of the polling based system that is currently in place, as they had heard it was an event based system and

almost expected events to be in place. This was most likely a confusion about the events going between the physical and digital devices from our initial explanation.

In terms of recommending the framework to colleagues or work more on their own with it, they could totally see that happening, but to make this easier, they did wanted some changes. One of them wanted more devices to show the extremes of what's possible with the framework, another wanted less devices to keep it simple enough where you could use it to teach programming in school, another wanted a way to simulate everything in Unity.

7.4.4 System usability scale

An SUS form gives a general measurement on how usable a system is. A modified version of the SUS form, to fit the unity framework, was given to the participants. Table 7.2 shows the results from the SUS form for the four participants in this round of workshops.

Participant	**SUS Score**
Student 1	80
Student 2	70
Developer 1	72.5
Developer 2	82.5
Average	76.25

Table 7.2: SUS score from workshop 1.

7.5 Results workshop 2

In this section, the results from the first workshop, where we had two participants split separately, are detailed. We will look at some of our observations during the workshop, the code from the participants and how they decided to solve the task, some information we got from the group interviews, and the results of the SUS.

7.5.1 Observations: Problems and challenges

As with the first workshop, given the hands-off approach to the task solving part of the workshop, the participants had to solve any software issues of their implementations on their own.

For this second workshop, there weren't any problems for us to solve for the participants given the restraints we gave on both ourselves and them, aside from the IP-change issue showing up once for one of the participants. The other participant did have some issues where we could see on the recording that he repeatedly closed and reopened Unity, where we thought there were issues with Unity itself, or hardware issues, but it was a pure code issue outside of the framework.

For the framework- and code-review part of the workshop, they were both mostly reading the wiki on the repository page, with some occasional review of the code inside the editor itself.

We do see that since this workshop only had two participants where they were working on their own, as compared to the first workshop where they worked in pairs, they didn't have any partners to discuss and get help from, but it didn't seem like that hindered them as they were both quick to get started with their implementations of the task, with both of them getting close to finishing it.

7.5.2 Code analysis

As with the first workshop, we told the participants that they could either start over, or expand upon what we had already made. Giving them the possibility to copy and paste code if they needed, almost encouraging it.

While it was interesting to see the different approaches to the task in the first workshop, it's also interesting to see that both participants here, did the same as one of the groups in the first workshop. They were both expanding on one of the examples, and at that expanding the Keep the Light alive example where we gave them two tile classes with game logic in them together with the game logic class. While the two participants this round had different results in terms of how far they got, what they coded was relatively similar.

7.5.3 Group interviews

As with the other workshop, we held a semi-structured group interview with both of the participants to get more concrete feedback on what was good and bad about the framework. As with the group interview section for the first workshop, we will abstain from quoting what the participants said as it wouldn't necessarily be accurate with a translation.

Usefulness

For the usefulness aspect on its own, we were interested in whether they could see themselves using the framework either in their jobs or some form of projects.

Both students said the framework was interesting and that they could see themselves using it in a project at a later time. One of them also brought up the point that was brought up by one of the game developers the first workshop about using the framework in an educational setting in schools.

Ease-of-use

For the usability questions, we were interested in the general aspect of whether the framework was easy to use. What was easy, difficult and what needed changes. Differing in this workshop from the first workshop, is that none of the participants had prior knowledge of the code before coming in.

Both participants said the wiki was good and understandable making it relatively easy to understand the framework. Noting also how the framework is simple and lightweight, but very flexible in terms of what could be done with it.

They did mention some issues in terms of method names, where they weren't sure what a method would return or do just based off the name itself, and had to think twice before using them. However, once they had the full picture on how the framework was set up, it was easier to understand what methods did and how things were connected.

Fit with context of use

For this part the we wanted to know how the framework would fit into how they as developers work when designing games.

Both of them weren't sure about how the framework fit a game development work flow, considering they were both students and at most had experience from that one game design course, and most of what they had done weren't related to this type of project. They did say it depended on the type of project, and they would have to see from there.

From the work flow question, we had posed a leading question on what they thought of the component structure, given what we had heard in the first framework to see if they agreed or not. They both agreed that a more Unity-like component system would be better for the general Unity developer, even if a software developer/computer science student would understand the current system just fine.

Summary

For this part of the interview we wanted to know some of the best and worst of the experience with working with the framework, or just about the framework itself, and whether they could see themselves recommending the framework to a colleague and any changes needed for that to happen.

The best part of the framework according to them was that it reminded them a bit of Arduino programming with setting LEDs, etc. It was also well structured so you knew what you were doing and didn't just pull a magical method out of nowhere.

For the worst part of the framework, one of them only had some small issues with how the naming convention we ended up using didn't properly fit C#'s naming conventions. The other had some issues that were more technical in terms of physical devices taking some time to connect and some delays making it more difficult to debug. However, this is a bit outside of what we were testing as we're focusing on the software.

In terms of recommending the framework to others, one of them brought up how his younger siblings in secondary school have started with Arduino programming and some Unity as well, but for Unity they've mostly done visual scripting. He could see high school students being a good target for this sort of framework.

7.5.4 System usability scale

An SUS form gives a general measurement on how usable a system is. A modified version of the SUS form, to fit the unity framework, was given to the participants. Table 7.3 shows the results from the SUS form for the two participants in this round of workshops.

Participant	**SUS Score**
Student 1	65
Student 2	87.5
Average	76.25

Table 7.3: SUS score from workshop 2.

Chapter 8

Requirements Update

8.1 Summary of Semi-structured Interviews

In this section we will summarize the participants' thoughts from the two workshops detailed in chapter 7. Every question will be listed with a summarized and short version of what everyone said. We will also cover a question regarding open source that wasn't included in the interview guide.

- *Do you think you could use this framework in a project? Explain.*
  - Depends on the project from a customer, and other ideas.
  - Could use it in Hobby projects.
  - Potential for use in an educational setting.
- *Was the framework easy to use?*
  - Easy to use after some exploration.
  - Not being Unity-based made it harder to get into.
  - Wiki helped in understanding the framework.
- *What was easy? Examples.*
  - Simple framework, thus easy to use.
  - Use of components.
  - General use with the lack of worry about network components.
- *What was difficult? Examples.*
  - Separation between game object and class object.
- *What could have been done differently? Examples.*

  - More documentation.
  - Unity-like code structure.
  - Higher abstractation.

- *How does the framework fit with how you work when you design games? Explain.*
  - Developers wanted a more Unity-like component system.
- *The best part about the framework.*
  - Felt like Arduino programming.
  - Easy to understand and start using.
  - Freedom and flexibility.
- *The worst part about the framework.*
  - Hardware: Connection time and delays.
  - Documentation.
  - Lack of Unity-like event system.
  - Examples structure. One wanted a different type of examples.
- *Would you recommend this framework to your colleagues?*
  - Could be recommended in use for High School teaching.
  - General consensus on agreeing that they could recommend the framework.
- *If no: What would need to change for you to recommend it?*
  - Not directly a no: Wanted to show extremes in terms of devices being able to connect to the system.
  - Not directly a no: The ability to simulate inside Unity.

8.2 Updated Requirements

In this section we will list the updated requirements for the framework based on feedback from the workshops. This will not include every change the participants from the workshops wanted, but there were a few good suggestions for changes.

The problem we encountered is that most of the changes the participants in the workshops wanted were code-base related and framework documentation. These won't really change the functional requirements, or the criteria from Johansen (2018) much, but the little that changes will be included. The changes included are in bold in table 8.1.

Criteria	Description
C1.0 - Reduce implementation cost	The framework and the technology used should reduce the overall cost of prototyping pervasive exergames, **by having sensible examples to work with to learn the framework and reduce learning time.**
C2.1 - IoT Flexibility	IoT technology should support for different I/O modules to allow developers to customize the game objects to their game design.
C2.2 - Addressability	IoT technology should provide the ability to uniquely identify and address the device.
C2.3 - Device-to-Device communication	IoT technology should support the ability to transfer and receive data from different IoT devices.
C2.4 - Distributed vs Local use	IoT technology should support both local and distributed applications.
C2.5 - Scalability	The IoT technology should support the ability to be used in pervasive games with different amounts or sorts of devices.
C3.1 - Interoperability	The framework and the technology used should support communication between various IoT devices.
C3.2 - Connecting new devices	The framework should provide intuitive handling of connecting new devices both during development and run time.
C3.3 - IoT Flexibility	The framework should allow developers to create custom interaction and visualization of different virtual I/O components that mirror traditional physical I/O modules.
C3.4 - Game logic centralized	The framework should support running the game logic outside of the IoT devices.
C3.5 - Framework	**Seamless integration with the development environment.**

Table 8.1: Updated criteria from Johansen (2018)

While the functional requirements won't change, the quality requirements do have some changes. However, the only quality requirements being changed here are performance requirements P1 and P2. The changes in the requirements can be seen in the second column of both tables 8.2 and 8.3. Both had a response measure of "Expected 40 messages per second", which is now changed to expect more than 60 messages per second. While this is probably the most desirable performance goal as the standard update speed of Unity

is 60 frames per second, or about every 17 milliseconds, a lower message rate could be enforced given a smooth rate up date in Unity. Though, this is heavily related to the hardware used in conjunction to the framework, meaning it's a bit outside our focus being the framework and software, but it can also possibly be enough with different variable handling on the Arduinos and some slight changes to the Unity framework. Lastly, a new criteria has been added for the framework to cover the wanted support for seamless integration with Unity, or the code-base change to a more Unity-based solution.

ID:	P1
Source:	Unity application
Stimulus:	Receives data from client
Artifacts:	Arduino device
Environment:	Normal operation
Response:	Data is used to update virtual objects
Response Measure:	**Expected more than 60 messages per second**

Table 8.2: Performance 1

ID:	P2
Source:	Unity application
Stimulus:	Send data to client
Artifacts:	Arduino devices
Environment:	Normal operation
Response:	Data is used to update and operate physical objects
Response Measure:	**Expected more than 60 messages per second**

Table 8.3: Performance 2

Chapter 9

Research Discussion

This section contains a discussion of the results derived from this study in relation to the validity of the research methods. The validity will be discussed in relation to objectivity, external validity, ecological validity and triangulation, all of which are described in Chapter 3.3.

9.1 Validity

9.1.1 Objectivity

In our study, much of the results were based on the semi-structured group interviews, and there was a risk that the interviewer could influence the participants' answers. To avoid this, audio and video recordings were taken during the user tests and interviews. This made it possible to analyze whether the participants were affected in their behavior after the interview. However, it is not possible to eliminate the influence of the interviewer completely. It seemed that most of the participants were sincere in their answers, although there is a possibility that some of the participants wanted to be positive in their answers.

9.1.2 External validity

There is sufficient basis for the conclusions drawn from the results of the data collection from the user tests to be valid. As mentioned in chapter 3.3, the criteria for research results being valid include transferability to other users. If there are too few participants, conclusions cannot be a representative generalization of the entire user group. Ideally, we should have ran a third workshop, but Nielsen and Landauer (1993) states that testing on 5 people is sufficient to reveal 85% of the errors. Should we run several workshops this would be at the cost of time, which we didn't have.

9.1.3 Ecological validity

NTNU UX-lab was used during the usability tests. This lab was set up to achieve high ecological validity. At the UX-lab, the test subjects were separated by a light wall, see figure 7.1. This meant that during the first workshop with group testing, it was possible to hear discussions through the wall. This UX-lab layout is not entirely optimal, and the results could have been different if the groups were more separated. When it comes to ecological validity, our setup is representing a certain degree of working in an open plan office.

9.1.4 Triangulation

In order to answer research questions 1, it was based on previous work and a literature study. To meet the triangulation requirement, there might have been added an additional method to increase the validity of the findings. Despite that, we saw it sufficient with previous work and literature study as this was intended as a first iteration in the design process of the framework.

For research question 2 and 3 it was based on a combination of literature review and design and creation. For better triangulation we could have added survey or field study for gathering a better insight into how game designers work against game engines. Here we could possibly have anticipated some of the feedback that came from the group interviews.

To answer research questions 4 and 5, a usability test with observation, interview and a survey was done to collect data. This meets the requirements for triangulation where there must be at least two methods for collecting data.

9.1.5 Test subjects

Recruiting the right participants is crucial to getting valuable results from a usability test. Our participants were primarily recruited through a presentation we had at Work Work, in addition to sending out an e-mail to students who had taken the subject Game Design (TPD4168 NTNU). We wanted to gather a group of representative end-users where the skills were from novice to experts, to improve the chances of meeting the true end-users. Norman (2013). For the usability test results to be valid, it was important that the users had not seen or tried the system in advance. Here we made a small mistake, where one of the participants during the presentation at Work Work wanted a link to the repository of the code (GitHub). Without us being able to confirm or deny whether the participant has read or tested the code in advance, we should have avoided this. Another thing that is important in recruiting is acquaintance. Often people you know can give a more positive opinion about the product they are testing, leading to weaker credibility in the answers. In our usability test there was none of the candidates we had significant relation to. We therefore assume that this did not affect the results we received.

9.1.6 The task

Moran (2018) explains in her article that a set of carefully crafted tasks is needed to succeed in any form of usability testing. There are some characteristics we have focused on in our task, as the usability test is primarily qualitative. We wanted to discover issues in the user's experience of the framework, and therefore aimed to create a task that would highlight the difficulties that the participants might experience. It was important to make the task as open and realistic as possible, as well as emotionally neutral. It contains no direct clues, but presents a hint that the example code from Github can be used as an starting point. This example code was used during the introduction of the task to create motivation. In conclusion, we left the task open for the participants to use their creativity to build on, if there was time left. In practice, the task was somewhat more time-consuming than assumed and none of the participants finished within the set time frame.

9.1.7 Researcher Bias

The area where researcher bias would affect the most, is the qualitative data gathering and analysis for the workshops, described in chapter 7. Oates (2012) describes the bias in relation to qualitative data analysis as how the researchers interpret the data is closely tied to the researcher, their identity, background, assumptions and beliefs. And that conclusions must then be more tentative than with quantitative analysis. Any preparations done before the data gathering could also affect the data collected. Including for us then the preparation of the workshops and the execution of them. As the workshops were the first iteration, the focus is rather on the execution of the workshops and the interviews at the end.

It's important to note that our work is from a developer perspective. This has some positives and negatives. The positive here is mainly that we understand our test users and can better connect with them. The negative is mainly that we can get blinded in terms of what we're working on without understanding the other perspectives, like a business perspective, project lead perspective, etc.

To facilitate any researcher bias during the workshops, and to stay as objective as possible, we made sure to keep them as similar as possible across both days. We had a plan for the general flow of the workshop, and when the participants worked on the task we gave them, as described in chapter 7, we were hands-off with the users with being in the observation room to not influence them in any way. The interviews were a bit more free-form, but they still followed a general guide on what questions to ask the participants, and were kept fairly neutral and similar across both days. However, there was one point in the second workshop interview where one of us asked a leading question based on what one of the participants said in the first workshop. This was clearly a biased question to probe and expect an answer from the participants, but it also got the participants to reflect about other things for the framework.

9.2 Alternative Research Approaches

9.2.1 Field Study

An alternative research method that could have provided good effect is the field study. A field study is a general research method that involves collecting data material in a specific field of research. It involves interviews and observations. For example, visiting Work Work to observe how they work on game design against Unity could provide us with a better basis for understanding and specifying the context of use. It is a normal approach in the Human-Centered Design process and could have been done before we started specifying the user requirements. See figure 3.4 *Understand and specify the context of use*. We thought that literature review together with the findings from Johansen's master thesis would be sufficient for a first iteration. A field study would have been at the expense of the time we had for the development of the framework and the usability test.

9.2.2 Field Experiment

A field experiment allows researchers to test theories and answer questions with a higher degree of external validity as it is done in a real-world environment. If we wanted to run an experiment in the end user's environment, it would involve doing the usability test at Work Work. Some advantages is that more participants with different backgrounds, might have brought different feedback. The product could have been tested in several creative ways, and maybe even on their own projects where they found it useful. For our part, it was important to have the opportunity to record audio and video to analyze the results afterwards. This could have been more difficult to archive if the experiment was held at Work Work. A field experiment might be better suited to the product at a later iteration of the design process. First iteration is very vulnerable to functional errors and it would be easier to handle these under the test environment at the university.

9.2.3 Open Source

An open source research approach would allow us to test how the project evolves by having other developers submit whatever changes they see fit for the framework. To explore this approach, we would have to have a working base early as a starting point. Going through with this approach would mean we would also have to curate any incoming changes from other developers, but it would allow us to see exactly what kind of changes users wanted. However, doing this approach would assume the people contributing to the framework had the necessary hardware to test making something, which could, given the flexibility of the framework, mean anything supporting MQTT. Another problem is that this approach would take an uncertain amount of time. Given that we could have uploaded the project and released it as early as the start of the thesis in January, how well the framework would do as an open source project would depend on who found it, or what kind of people we would manage to get in contact with. To have time to try this approach, we would have had to do this on the first iteration of the framework, which most likely would have been to early.

Chapter 10

Discussion

10.1 Research Question 1

What are the requirements for an open source architecture to support the development of pervasive exergames?

In order to answer research question 1, we chose to look at the criteria from Johansen thesis, as well as literature review. The criteria we considered most relevant are presented in table 4.1. Of these criteria, we chose to change C3.1 (C3.2 in Johansen's table). We removed the part there various IoT devices could communicate without knowing the specification of each device. We considered this a smart functionality, but demanding to implement as not all potential IoT devices have a configuration they can send to the system. The change here is that Unity will know the specification of the connected devices. Unlike Moto Tiles, we have criteria that state that our framework should be open source and support customizable IoT devices with various sensors. This opens the possibility of creating exergames, where the devices are different both physically and digital. To better support open source, we decided that the game logic should be centralized. This would allow the framework with some simple adjustment to work with alternative game engines and IoT devices. An important requirement that must be made to the framework is the choice of message protocol. It should have good credibility, be lightweight and fast for streaming sensor data to low-energy chip sets. To lower the barriers of using the framework, we chose a message protocol that was free and defined as an ISO standard.

10.2 Research Question 2

What existing software technologies are best suited to realize these requirements?

Based on the research we have done; we have concluded that technologies that are either well established or defined as a standard are the technologies that will live the longest.

In other words, it is wise to choose technologies where chances are greater that they will be supported in the future. In addition to this, it is necessary to look at which of these technologies will meet the requirements of sensor based IoT devices.

As a message protocol we have chosen MQTT. It was chosen because it's an ISO standard and were designed for remote devices where a small code footprint and limited network bandwidth are important.

For the game engine, there was a project requirement that Unity should be the chosen one. Regardless of this requirement, Unity is one of the largest gaming engine with a broad platform coverage, it supports third party frameworks such as MQTT and allows you to develop games for free as long as gaming revenue does not exceed $ 100K per year. Thus, this game engine is well suited to realize the requirements for the development of pervasive exergames with a tangible user interface. Other game engines, like Unreal Engine, could have been chosen. A fully open source game engine, like Godot, could have been used for a fully Open Source solution, if desired.

When it comes to the IoT devices, we chose Arduino as a technology platform. Arduino is an open-source platform where the focus is on user-friendly hardware and software. It uses C ++ as a programming language and has good documentation in addition to a large environment of users within IoT projects. Regarding the design and creation of IoT devices, there are lots of cheap SoC microcontrollers, and sensors based on Arduino. We therefore considered it a favorable technology platform for realizing pervasive exergames.

Between the IoT devices and the machine running Unity, we have chosen to place a Raspberry PI. This runs a linux distro where the software "dnsmasq" is used for a DHCP server, “hostapd” for an access point and Mosquitto MQTT as a message broker. All three of these softwares are built on open source and are well established within the Linux community. For the development of the framework, we saw it as user-friendly to have this raspberry pi as a hub for connecting the IoT devices and for handling MQTT messages. In theory, this unit can be replaced by the same machine running Unity.

10.3 Research Question 3

What are the challenges of implementing such an architecture for a specific target technology?

To answer this question, we need to look at the challenges we had during the Design and Creation progress, with tie-ins to the usability testing. In chapter 6, we detailed the implementation choices we made for the architecture, and how different parts of the architecture works. Through this implementation process, we got some first-hand experience in the challenges involved. These include uncertainties with the technologies used, and how to use said technologies.

One of the biggest issues we noticed that most people will probably run into is unfamiliarity with libraries used on the Arduino side. This was apparent to us when we tried develop the Arduinos using their functionality. Most of the this came down to handling the data from the components using the libraries. To alleviate some of the unfamiliarity, it was

helpful to use the examples shipped with the libraries as a base to work from. This was especially helpful with the MQTT library used as it set the base for all the Arduino device configurations. This unfamiliarity also stemmed from the inexperience with using C++ as a programming language, which was less of an issue when dealing with Unity, as C# is relatively similar in syntax to Java, which we have experience with. This meant we had to learn how the libraries worked, how Arduinos worked in its execution flow, and how specific things worked in C++.

For developing the Unity-side of the architecture, something didn't occur to us until we had it tested with users, which is how specific you want to be towards the technology. Going into this project, we didn't have much experience with developing for Unity and had more general programming knowledge. Given this we ended up going for a more general C# solution, than a more Unity focused solution. This has its pros and cons depending on how you weigh them. Given the more general C# solution, porting the framework to a different game engine using C# should be easier than if we had a more Unity focused solution. However, given the fact that our target demographic is Unity developers, we should have considered this sooner and made a framework more suited for Unity. It's easy to fall into pitfalls depending on the mindset you have as you begin developing. This can be seen in the results from our testing in chapter 7.

While the focus of our work is related to software, we can't ignore the challenges we had with the hardware. Our initial prototypes were made using breadboards to connect the components with the Arduinos. The components themselves weren't incompatible with the Arduinos as they worked fine in the soldered prototypes, the tiles, but there were a lot of issues with the components not being detected, or suddenly not working. We think this could be related to the slight physical connection errors in the breadboards, as several were tested and the setups could stop working without even touching them.

These issues somewhat limited our exploration with different components as the goal was to use the ones we had in the red and blue tiles. With the time constraints we had, this led to limitations on what kind of prototypes we had to work with. While we could influence what prototypes were made, we weren't directly in control of that process. This somewhat limited what components were coded for in Unity, but this made it all the more important to make sure that it's possible to extend on this. This also limited the examples made to some extent.

10.4 Research Question 4

How do potential users of the resulting framework asses its usefulness and usability?

For this question, we need to consider what potential users we tested the framework with. Our user group consists of game developers with programming experience using Unity. Of the people who participated during the testing, we had two game developers and four students who had a game design course at the university. None of them knew much about pervasive games, and had not developed for that before, which may influence some of the answers they gave or the results we ended up with. The usability test focused only

on how the framework was towards game development in Unity. We do not test how to program IoT devices according to the Arduino side of it.

For the usefulness of the framework, all of the participants said the framework could be used in projects, or IoT projects outside work, given the idea and how the framework could fit. One of the things that came up during both interviews was the idea of using the framework in a school setting to teach teens coding, as the framework is relatively simple and reminded some of them of Arduino programming, which is what one of the participants' brother had done at school already. Another example brought up was a football setting where the athletes could run between boxes similar to how Follow the Light works, which is more of a game idea than a use of framework idea, but still interesting to look at.

From the aspect of usefulness, it's interesting to see that the participants could see themselves using the framework, and that they already thought of ideas for other places it could be used as well.

On the aspect of usability, we had SUS forms the participants filled in. While there might be some uncertainties regarding the score with how the participants answer it, where participants can give more positive answers to be nice, it gives a good guideline on where we are with the framework. With the SUS score average for a good usability set to 68, our average score across all participants being 76.25 means we're on the right track towards a good framework.

However, there were still changes the participants wanted to see that could improve the system more. As they were Unity developers, they might be more biased towards that as a development tool, and as such might explain their desired changes for a more Unity-focused framework. But we can see this being better for general Unity developers, to have tools more integrated to the working environment they are used to.

The participants also had some issues in the separation of objects in Unity referring to the physical objects and the game objects in Unity, and the general link between physical objects and digital twins. We could see this being an issue as the documentation on the system both in code and on the wiki pages for the Github repository weren't completed yet. But what we had helped fill some of the holes, and the participants still saw the framework as relatively small, but still very flexible.

10.5 Research Question 5

Based on the previous questions; what are the requirements for such an architecture?

An updated requirement is that the framework must have a good documented wiki page, where all functions must be clearly explained. In addition, good code examples must be enclosed so that a developer can more easily familiarize himself with the framework. This was a requirement already from earlier, but there was a desire for an improvement of the existing documentation.

The participants wanted the framework to remain open source so that they could use it as a free tool for their own projects and get the opportunity to contribute to the framework.

It was not discussed which software license the framework should have beyond that it remains open source. It is therefore not possible to conclude any updated requirements on the framework in connection with which license the framework should be submitted to.

All participants in the usability test were recruited following a requirement for experience in Unity game development. One common idea among the participants was to make the code base even more aimed at Unity. Those of the participants with a heavier programming background liked that the framework was small and clear, it was positive that it wasn't big and complicated.

An interesting thought is; What results would we have if we had two tasks, one aimed at Unity and one against another game engine. Would there be an equally great desire for a framework dedicated to the individual game engine or not? Since the EXACT project made demands on Unity as a gaming engine, a new requirement for this framework would be to change the code base to be more focused on Unity.

Chapter 11

Conclusion

The motivation for our study was to design and create a user-friendly framework that will make it easier for game developers to create pervasive exergames. The framework will give developers the opportunity to focus on a tangible user interface, with the ability to connect customizable IoT devices to a digital twin object in Unity. The study has focused on design and creation with an experiment in the end, where we test the usability of the framework. We hope that this framework will be an important part of the EXACT project, and will open the possibility for new pervasive exergames to be developed.

Five research questions were designed to conduct the research of this study. The first research question was to find out what requirements a framework should have to support the development of pervasive exergames. Based on a literature study and the findings in Johansen's Master's thesis, it was concluded that the requirements for the framework should be based on the criteria of Johansen's thesis and the extent to which the literature study agrees with it. We therefore ended up with the requirements presented in table 4.1.

The second research question regarded which existing technologies are best suited to realize the requirements we found in research questions 1. The technologies we chose were; MQTT as a message protocol between the IoT device and the gaming engine, Arduino as a technology platform for the development of IoT devices and a Raspberry Pi running the operating system Raspbian with DHCP, Access Point and MQTT Broker services. The game engine was already decided to be Unity before we defined this research question. It's important to stress that everything regarding the Raspberry Pi in terms of the DHCP and access point is not part of the actual framework developed, but more of a component to help the communication between devices.

The third research question regarded the challenges in the implementation of such a framework for the specific technologies chosen. The biggest challenge we thought we would face would be the libraries used for the Arduinos and the unfamiliarity with the programming language C++. What turned out to be the biggest frustration during prototyping and testing in the early stages ended up being the breadboard setups before we had physical tiles to work with. The connections between physical components on the breadboards

were unstable until we had a soldered solution. The direction you take your code is also a challenge as you may end up going varying directions depending on how you as a developer first think the code should be structured, which may turn out to need change after testing.

The fourth research question regarded how the potential users of the resulting framework asses its usefulness and usability. For this we had semi-structured interviews and SUS forms for the participants of the workshops we held. For usefulness some of them could see themselves using it in projects, both for work on a hobby basis, and some could see it being used in education. The discussion during the interviews lead to the participants thinking about various exergame ideas that could be developed with the framework. For usability the SUS form revealed that we had a score of 76.25 which is considered "good". Bangor et al. (2009). There was an agreement from the participants between both workshops that the framework should be more integrated into Unity. We're pleased with the results from the usability tests considering that this is the first iteration of the framework.

The last research question regarded what would be the requirements for this architecture based on the previous research questions. During the workshops we got a confirmation that the documentation in terms of the wiki-pages including the code examples were good, but there is room for improvement. The participants also agreed that the framework should remain open source as the code being open source had an impact on the decision making in terms of what frameworks to use. There was also a consensus that the code-base should be more aimed at Unity if the expected end-user is a Unity developer.

11.1 Recommendations for Further Work

The results of the research indicated that there are several improvements for future development on the framework. As a start, the code should be ported from the more general C# solution to a more integrated Unity solution, which means it should be more familiar to the way Unity handles game objects and class components.

One thing that was not addressed during the usability test, but is important for future work, is to look deeper into the security of the framework. In particular, it is important to add extra care in making the broker and the communication secure.

With the Unity integrated solution, one idea to further develop the framework, would be to create game objects representing the physical components, as an example a sphere for an LED. This could enable the end user to use these components as building blocks to create a digital twin by nesting them in the Unity Editor.

Regarding the Arduino code, there are several structure changes that could be done to make it easier for future contributors. This could involve having the functionality for each Arduino component in their own classes, and thus provide a better API.

We consider the current version of the framework more akin to an alpha version, and as such there are several steps needed to be fulfilled before a proper beta version is ready, among these the changes described above. Preferably more iterations of testing need to be done after the changes regarding the Unity integration mentioned above. In these extra iterations, it could be wise to do a field experiment to further test the framework outside of a

more controlled environment like the UX-Lab at NTNU. This could spark an interest among potential contributors and users as we see the project being released as an open source project.

Bibliography

Adafruit, 2018. Adafruit feather huzzah with esp8266 - loose headers. https://www.adafruit.com/product/2821, accessed on 2018-10-29.

Ashton, K., et al., 2009. That 'internet of things' thing. RFID journal 22 (7), 97–114.

Bangor, A., Kortum, P., Miller, J., 2009. Determining what individual sus scores mean: Adding an adjective rating scale. Journal of usability studies 4 (3), 114–123.

Bass, L., Clements, P., Kazman, R., 2003. Software architecture in practice. Addison-Wesley Professional.

BCS, T. C. I. F. I., 2017. The Internet of Things: Living in a connected world. British Computer Society.

Buxmann, P., Diefenbach, H., Hess, T., 2012. The software industry: Economic principles, strategies, perspectives. Springer Science & Business Media.

Chen, L., Babar, M. A., Nuseibeh, B., 2013. Characterizing architecturally significant requirements. IEEE software 30 (2), 38–45.

CORBA, 2018. Common object request broker architecture. http://www.corba.org/, accessed on 2018-10-30.

Crytek, 2018. Cryengine. https://www.cryengine.com/.

Davis, F. D., 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, 319–340.

Dickinson, A., Lochrie, M., Egglestone, P., 2015. Ukko: enriching persuasive location based games with environmental sensor data. In: Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play. ACM, pp. 493–498.

Ecclipse, 2018. An open source mqtt broker. https://mosquitto.org/, accessed on 2018-11-05.

EpicGames, 2018a. Epic games. https://www.epicgames.com/site/en-US/home.

EpicGames, 2018b. What is unreal engine 4. https://www.unrealengine.com/en-US/what-is-unreal-engine-4, accessed 08-12-2018.

Foundation, R. P., 2018a. Raspberry pi 3 model b. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/, accessed on 2018-10-29.

Foundation, R. P., 2018b. Raspbian. https://www.raspbian.org, accessed on 2018-10-31.

Gaff, B. M., Ploussios, G. J., 2012. Open source software. Computer 45 (6), 9–11.

Glaessgen, E., Stargel, D., 2012. The digital twin paradigm for future nasa and us air force vehicles. In: 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA. p. 1818.

Godot, 2018. Godot engine - free and open source 2d and 3d game engine. https://godotengine.org/.

Grieves, M., Vickers, J., 2017. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In: Transdisciplinary perspectives on complex systems. Springer, pp. 85–113.

Henry, J. M., 2018. The development of a modular framework for serious games and the internet of things. Ph.D. thesis, Liverpool John Moores University.

Houde, S., Hill, C., 1997. What do prototypes prototype? In: Handbook of human-computer interaction. Elsevier, pp. 367–381.

Intel, Jun 2016. Choose the right communication pattern for your iot project. https://software.intel.com/en-us/articles/communication-patterns-for-the-internet-of-things, accessed on 2018-10-28.

ISO, Mar 2010. Ergonomics of human-system interaction – part 210: Human-centred design for interactive systems. https://www.iso.org/standard/52075.html, accessed on 2019-04-28.

ISO, Jun 2016. Information technology – message queuing telemetry transport (mqtt) v3.1.1. https://www.iso.org/standard/69466.html, accessed on 2018-10-30.

ISO, Mar 2018a. Ergonomics of human-system interaction – part 11: Usability: Definitions and concepts. https://www.iso.org/standard/63500.html, accessed on 2019-05-10.

ISO, 2018b. International organization for standardization. https://www.iso.org/home.html, accessed on 2018-10-30.

Jessen, J. D., Lund, H. H., 2016. Evaluation and understanding of Playware Technology–trials with playful balance training. Technical University of Denmark, Department of Electrical Engineering.

Johansen, P. B., 2018. Iot-based pervasive game framework - a proof of concept case study. Master's thesis, Norwegian University of Science and Technology, released on NTNU Open Access.

Kathrin, u., 2015. How fast can you publish mqtt messages? | bitreactive$_2$015. URL

Konstantinidis, E. I., Bamparopoulos, G., Bamidis, P. D., 2015. Transfering full body exergames from desktop applications to mobile devices: the role of the internet of things. In: Interactive Mobile Communication Technologies and Learning (IMCL), 2015 International Conference on. IEEE, pp. 254–258.

Konstantinidis, E. I., Billis, A. S., Paraskevopoulos, I. T., Bamidis, P. D., 2017. The interplay between iot and serious games towards personalised healthcare. In: 2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games). IEEE, pp. 249–252.

Kosmides, P., Demestichas, K., Adamopoulou, E., Koutsouris, N., Oikonomidis, Y., De Luca, V., 2018. Inlife: Combining real life with serious games using iot. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, pp. 1–7.

Manley, A. F., 1996. Physical activity and health: A report of the Surgeon General. Diane Publishing.

Martin, K. A., Laviola, J. J., 2016. The transreality interaction platform: Enabling interaction across physical and virtual reality. In: Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016 IEEE International Conference on. IEEE, pp. 177–186.

Mcleod, S., 2017. Qualitative vs quantitative research | simply psychology. `https://www.simplypsychology.org/qualitative-quantitative.html`

Microsoft, 2018. .net. `https://www.microsoft.com/net`, accessed on 2018-10-30.

Montola, M., Stenros, J., Waern, A., 2009. Pervasive games: theory and design. CRC Press.

Moran, K., 2018. Writing tasks for quantitative and qualitative usability studies. `https://www.nngroup.com/articles/test-tasks-quant-qualitative/`

Moto, 2017. Moto tiles. `http://www.moto-tiles.com`, accessed 02-12-2018.

Muir, S. P., 2005. An introduction to the open source software issue. Library Hi Tech 23 (4), 465–468.

Nielsen, J., Landauer, T. K., 1993. A mathematical model of the finding of usability problems. In: Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems. ACM, pp. 206–213.

Norman, D., 2013. The design of everyday things: Revised and expanded edition. Basic books.

Oates, B. J., 2005. Researching information systems and computing. Sage.

Oates, B. J., 2012. Researching information systems and computing. Sage, first published 2006.

Oh, Y., Yang, S., 2010. Defining exergames & exergaming. Proceedings of Meaningful Play, 1–17.

ORACLE, 2018. Enterprise javabeans technology. https://www.oracle.com/technetwork/java/index-jsp-140203.html, accessed on 2018-10-30.

Patten, J., Griffith, L., Ishii, H., 2000. A tangible interface for controlling robotic toys. In: Conference on Human Factors in Computing Systems: CHI'00 extended abstracts on Human factors in computing systems. Vol. 1. Citeseer, pp. 277–278.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T., 1994. Human-computer interaction. Addison-Wesley Longman Ltd.

Sauro, J., 2011. Measuring usability with the system usability scale (sus). https://measuringu.com/sus/

Siegel, S. R., Haddock, B. L., Dubois, A. M., Wilkin, L. D., 2009. Active video/arcade games (exergaming) and energy expenditure in college students. International journal of exercise science 2 (3), 165.

Skjæret, N., Nawaz, A., Morat, T., Schoene, D., Helbostad, J. L., Vereijken, B., 2016. Exercise and rehabilitation delivered through exergames in older adults: An integrative review of technologies, safety and efficacy. International journal of medical informatics 85 (1), 1–16.

Sommerville, I., 2016. Software engineering, 10th Edition. Pearson.

Sparkfun, 2018. Choosing an arduino for your project. https://learn.sparkfun.com/tutorials/choosing-an-arduino-for-your-project/all, accessed on 2018-10-29.

Svanæs, 2015. Ntnu health informatics usability and design lab. interactions 22 (3).

Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., Sui, F., 2018. Digital twin-driven product design, manufacturing and service with big data. The International Journal of Advanced Manufacturing Technology 94 (9-12), 3563–3576.

Tognazzini, B., 1992. Tog on interface, 1st Edition. Addison-Wesley.

Unity, 2018. Unity. https://unity3d.com/, accessed on 2018-11-01.

Walsham, G., 2006. Doing interpretive research. European journal of information systems 15 (3), 320–330.

Wiemeyer, J., Kliem, A., 2012. Serious games in prevention and rehabilitation—a new panacea for elderly people? European Review of Aging and Physical Activity 9 (1), 41.

Wortmann, F., Flüchter, K., 2015. Internet of things. Business Information Systems Engineering 57 (3), 221–224.

Appendix

Appendix A - Code

The MQTT implementation used in the framework was created by GitHub user vovacooper(`https://github.com/vovacooper/Unity3d_MQTT/`). Included below are a few of the implemented classes in the framework, namely the MQTTHandler, TwinObejct base class, DeviceComponent base class.

Folder Structure

The folder included, Project.zip, is devided into two subfolder. One containing Arduino code, called Arduino. The other containing the Unity project and code, called Unity. The Arduino folder stucture diagram contains all folders and files included, except the ".vscode" folders(Visual Studio Code editor program settings folder generated by the program). The Unity folder structure diagram will list project folders as "Project Folders/Files" and "Code files"(our code files) due to the amount of folders and files in the Unity folder. ".meta" files generated by Unity3D has been omitted as well.

Arduino Folder Structure

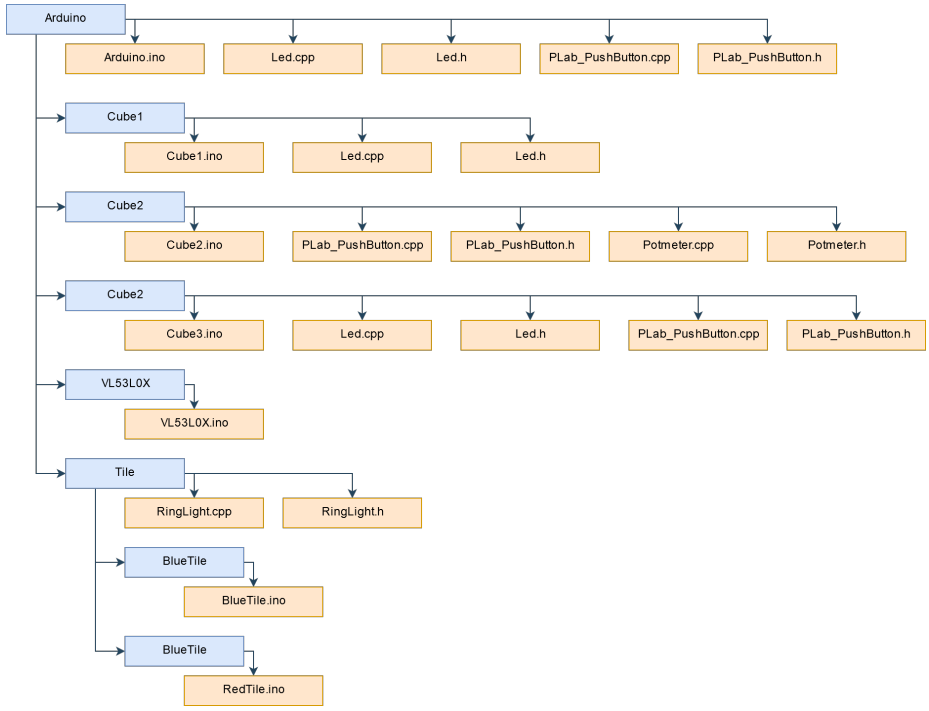


Figure 11.1: Project.zip -> Arduino folder diagram

Unity Folder Structure

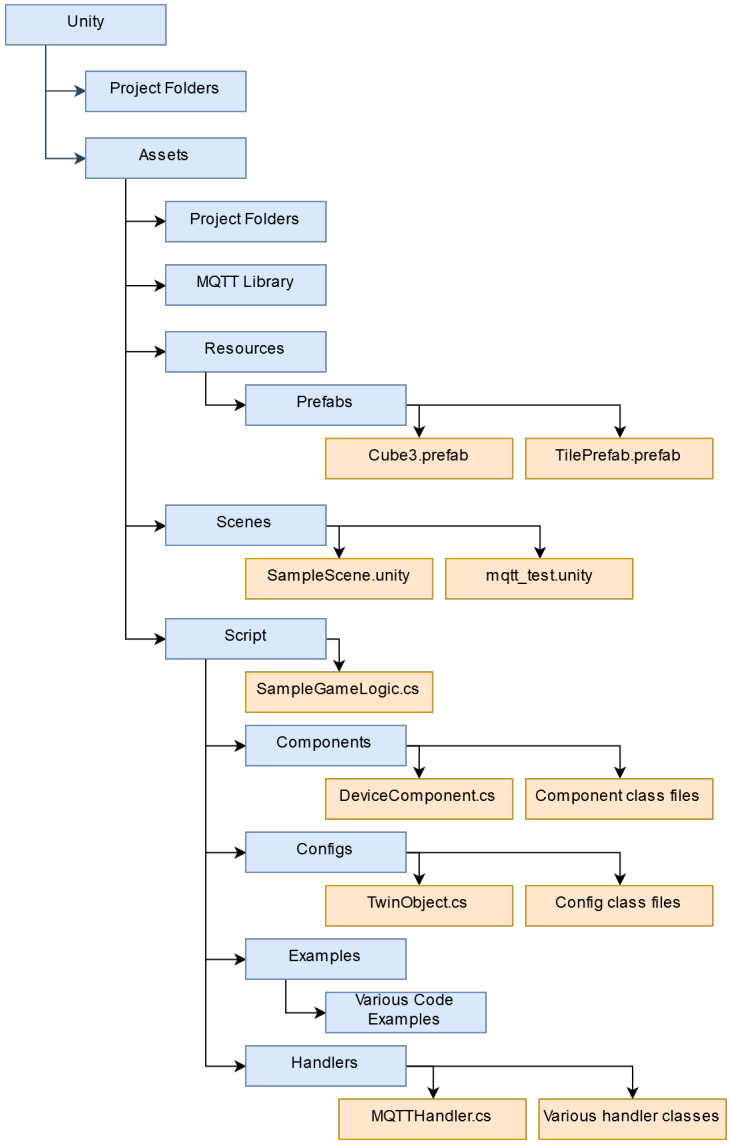


Figure 11.2: Project.zip -> Unity folder diagram

Appendix B - Documentation

1. Installation

This guide assumes you have Unity (Unity3D) already installed.

Unity

1. Download the git repository, either by cloning it or downloading as a zip.
2. Place the library folder your Unity project's Asset folder.

2. MQTTHandler

Fields:

Private:

MqttClient client MQTT Client Object from the MQTT library found at https://github.com/vovacooper/Unity3d_MQTT. Handles the connection to the MQTT Broker, and sending and receiving of messages. Message receiver runs on a second thread outside of the main unity thread.

List twinObjects Holds all the TwinObject objects set by the user. Also acts as a controller for the allowed device types to connect to the game. TwinObject explained in its section.

List msgBuffer Message buffer for incoming MQTT messages. Accessed by the update function ran within the main Unity thread. MessagePair explained in its section.

Methods:

MQTTHandler(string, int) - Constructor

Constructor. Takes hostname/ip-address of the broker and port. Sets up the MQTT Client and connects to it. Sets up the callback function for the incoming messages.

update() - Void

Update function to run through the mssage buffer filled by the callback function from the MQTT client. Calls the appropriate private methods to connect and update TwinObjects, and their components.

addTwinObject(TwinObject) - void

Takes one TwinObject and ads it to the list of TwinObjects. Sets the mqttHandler to the TwinObject for its mqttHandler-calls.

allDevicesConnected() - boolean

Checks all TwinObjects in the twinObject-list to see if they're connected to a physical device. Returns true if all devices are connected.

getConnectedTwinObjects() - List

Returns all the TwinObjects in the twinObject-list that have a connection to a physical device.

getTwinObjectList() - List

Returns the list of all TwinObjects, regardless of whether they are connected or not.

getObjectByID(string) - TwinObject

Returns the TwinObject with the given string ID if it exists in the TwinObject-list.

3. TwinObject

Abstract class.

Code for this class at (need link)

Parent Classes

MonoBehaviour

Child Classes

RedTile, BlueTile, Cube1, Cube2, Cube3, Tile

Fields

Private:

MQTTHandler mqttHandler

MQTTHandler object reference. Used to call the message sending function.

string deviceID

ID of the device. Set when a device with the appropriate configuration is connected and linked with the TwinObject. A unique identifier.

bool linked

If a physical device is connected to the TwinObject, this variable is true.

int pingCount, pingTime

Used in ping handling with the physical device.

Protected:

string configName

Configuration name of the TwinObject set by the child classes, and used as an identifier for connecting devices to set up links.

Appendix B - Documentation

Methods

getDeviceID() - string

Returns the unique device ID string of the object.

getLinkStatus() - bool

Returns true if the TwinObject is linked with a physical device.

setLinkStatus(bool) - void

Sets the link status of the TwinObject. Usually called by the MQTTHandler.

linkDevice(string) - void

Sets the deviceID string of the TwinObject to the device ID of the physical device. Set the link status bool to true and sends a ping message to the physical device.

getConfigName() - string

Returns a string with the name of the configuration set by the child class.

3.1 Red Tile

Parent Classes

MonoBehaviour, TwinObject

Fields

Protected:

RingLight ringLight

Object reference for the RingLight component class.

TonePlayer tonePlayer

Object reference for the TonePlayer component class.

IMU imu

Object reference for the IMU component class.

Methods

getRingLight() - RingLight

Returns the RingLight object for this device.

getTonePlayer() - TonePlayer

Returns the TonePlayer object for this device.

getIMU() - IMU

Returns the IMU object for this device.

3.2 Blue Tile

Parent Classes

MonoBehaviour, TwinObject

Fields

Protected:

RingLight ringLight

Object reference for the RingLight component class.

TonePlayer tonePlayer

Object reference for the TonePlayer component class.

IMU imu

Object reference for the IMU component class.

TimeOfFlight timeOfFlight

Object reference for the TimeOfFlight component class.

Methods

getRingLight() - RingLight

Returns the RingLight object for this device.

getTonePlayer() - TonePlayer

Returns the TonePlayer object for this device.

getIMU() - IMU

Returns the IMU object for this device.

getTimeOfFlight() - TimeOfFlight

Returns the TimeOfFlight object for this device.

4. DeviceComponent

Abstract class.

Child Classes

IMU, TimeOfFlight, RingLight, TonePlayer

Fields

Protected:

TwinObject device

Sets the device that uses the component class. Used for message sending to the physical version of the TwinObject when a component is updated, or for appropriate calls.

Methods

setDevice(TwinObject) - void

Sets the TwinObject device of the component.

getDevice() - TwinObject

Returns the TwinObject device of the component.

update() - abstract void

Called for some of the component classes inheriting this class.

Appendix B - Documentation

4.1 IMU

Parent Classes

DeviceComponent

Fields

Rotations rotation

Object to hold the rotation(link) of the device measured from the IMU.

bool tapped

Bool set when the IMU detects a tap on the physical device. Set to false each frame if set to trrue.

Methods

IMU(TwinObject) - Contructor

Sets the device of the component to the TwinObject. Sets a defult 0,0,0 rotation for the rotation object.

update() - void

Sets tapped bool back to false.

getRotation() - Rotation

Returns the Rotataion(link) object of this class.

setRotation(Rotation) - void*

Sets the rotation(link) object of this class to a different rotation object.

setRotation(float, float, float) - void

Sets the Rotation(link) object's yaw, pitch and roll values.

setTapped() - void

Sets the tapped bool to true. Usually called when the TwinObject receives a tapped call over MQTT from the physical device.

justTapped() - bool

Returns the tapped bool. Useful to see if the device was just tapped.

4.2 RingLight

Parent Classes

DeviceComponent

Fields

Private:

Color color

Unity Color object. Holds various color related values.

bool state

Bool value for whether the ring light is on or off.

int numOfLeds

Number or active leds on the ring light.

List ledList

List of RingLightLed(link) objects. Each objected represents one led on the physical ring light.

Methods

RingLight(TwinObject, Transform) - Constructor

Takes the TwinObject device to set for the DeviceComponent device object. The Unity3D Transform object is used to extract the RingLightLed's(link) from the 3D virutal representation of the RingLightLeds(link). If they don't exist, the led list is filled with dummy leds, that function the same way.

toggle() - void

Flips the state boolean. Calls the setState method with the flipped bool.

setState(bool) - void

Sets the state of the ring light. Sets the state of the RingLightLeds(link) as well according to the number of active leds.

getState() - bool

Returns the state bool of the ring light.

setColor(Color) - void

Sets the Color of the ring light. Sets the color of each individual led as well.

setColor(Color, int) - void

Sets the Color of an individual led in the led list.

setAllLedsColor() - void

Sends all the individual led colors down to the physical device.

getColor() - Color

Returns the Color varialble from the ring light.

setNumOfLeds(int) - void

Sets the active number of leds on the ring light.

getNumOfLeds() - int

Returns the number of active leds on the ring light.

getLedList() - RingLightLed[]

Returns the list of RingLightLeds(link) from the ring light.

4.3 TimeOfFlight

Parent Classes

DeviceComponent

Fields

Private:

int distance

Holds the distance measured by the TimeOfFligth component on the physical device.

Methods

TimeOfFlight(TwinObject) - Constructor

Sets the TwinObject device to the component device. Defaults distance to 0.

setDistance(int) - void

Sets the distance variable. Usually called by the TwinObject when a value is received over MQTT.

getDistance() - int

Returns the distance value.

4.4 TonePlayer

Parent Classes

DeviceComponent

Methods

TonePlayer(TwinObject) - Constructor

Sets the TwinObject device to the component device.

playTone(int) - void

Sends a play message over MQTT with a frequency. Plays a tone that lasts forever at the given frequency.

playTone(int, int) - void

Sends a play message over MQTT with a frequency and duration. Plays a tone with the frequency and duration.

stopTone() - void

Sends a stop message over MQTT. Stops whatever tone the tone player is playing.

Appendix C - Questionnaire

Questionnaire

Name:	
Age:	
Gender:	
Employee / Student:	
Working title / Study program:	

Your experience from (1-5) where 1 = very little and 5 = very much

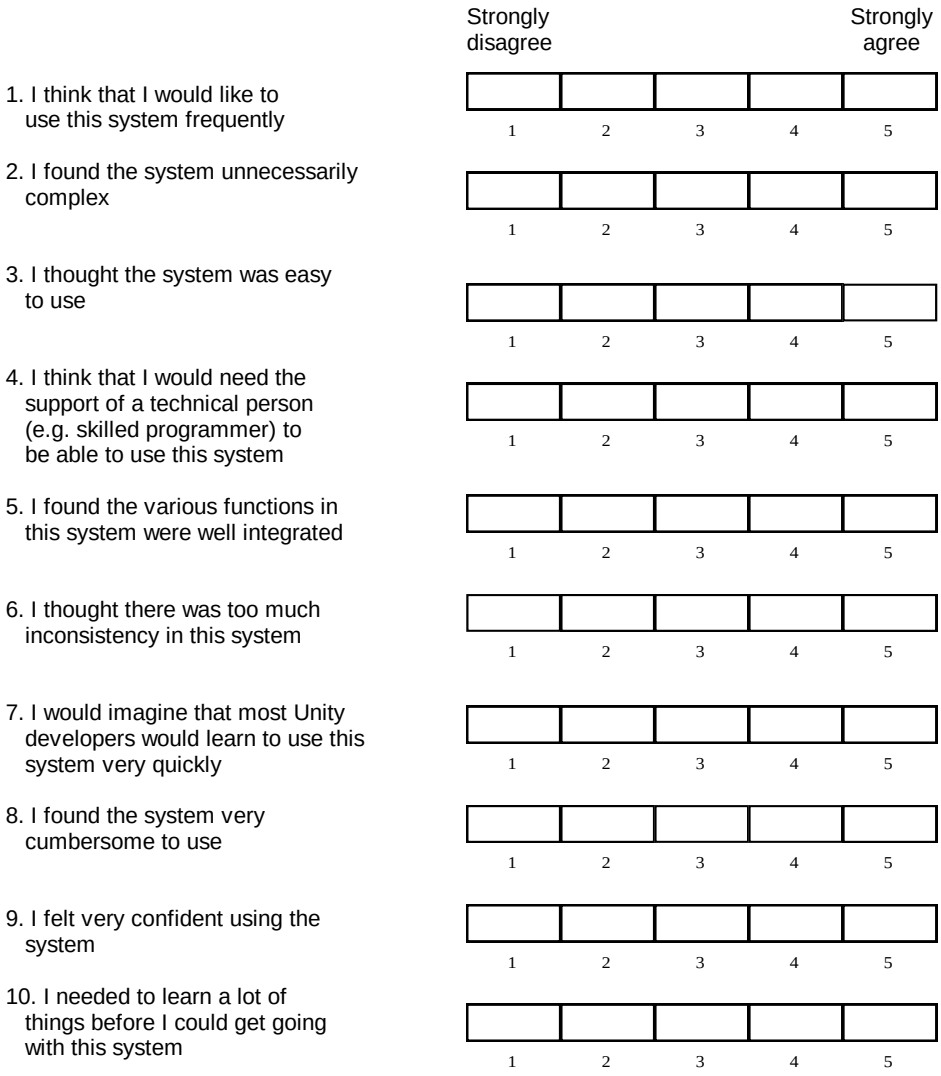
Unity:	1 2 3 4 5
C#:	1 2 3 4 5
Internet of Things	1 2 3 4 5
Use of frameworks:	1 2 3 4 5
Game design:	1 2 3 4 5
To what extent are you free to choose frameworks in projects:	1 2 3 4 5

Check of other platforms you have experience with in game development:

Windows:	
Mac:	
Linux:	
Playstation:	
Nintendo:	
XBOX:	
iOS:	
Android:	
Other:	

Appendix D - SUS form

System Usability Scale



Appendix E - Partner Agreement

Deltakelse i forskningsprosjekt.

Samtykkeerklæring

Dataspill har stort potensiale som hjelpemiddel i fysisk rehabilitering. I dette prosjektet skal vi evaluere programmeringsrammeverk for utvikling av slike spill.

Institutt for Datateknikk og Informatikk ved NTNU er ansvarlig for prosjektet, i samarbeid med Fakultet for Medisin og Helsevitenskap ved NTNU. Oppgaven befinner seg innen fagområdet Menneske-Maskin Interaksjon.

Formålet med dette konkrete prosjektet er å få konstruktiv tilbakemelding fra potensielle brukere av et slikt rammeverk. Vi vil samle data gjennom observasjon og uttesting av denne teknologien i lab. De data som samles er bilder, video og lyd. Datamaterialet planlegges anonymiseres ved prosjektslutt 31.12.2019.

Jeg har mottatt informasjon om studien, og fått anledning til å stille spørsmål. Jeg er klar over at det er frivillig å delta, og at jeg kan trekke meg fra studien når som helst uten å oppgi noen grunn.

Det vil bli tatt video- og lydopptak. Dette gjøres for at vi skal kunne analysere opptakene i etterkant og sikre at vi har forstått deres utsagn og handlinger riktig. Vi vil sørge for at materiale vil bli anonymisert slik at det ikke vil være mulig å føre opplysningene tilbake til enkeltpersonene som deltar i prosjektet.

Dette innbærer at informasjon som blir formidlet til offentligheten ikke vil kunne settes i sammenheng med den enkelte. Enkeltpersoner vil ikke kunne gjenkjennes i de endelige publikasjoner. Det er kun de involverte i prosjektet som vil kunne se opptakene i ettertid.

Jeg samtykker i å delta.

Trondheim, _____ (dato)

_____

Underskrift

Kontaktperson:

Professor Dag Svanæs
Institutt for Datateknologi og Informatikk
NTNU - Trondheim
e-mail: dags@idi.ntnu.no

Appendix F - Agenda for Workshop

Agenda Workshop 26 & 27. March

16.00 – 16.30	Introduction: - The purpose - Architecture - Demo - Consent form
16.30 – 17.00	Example review on Github
17.00 – 17.15	Introduction to the assignment
17.15 – 17.30	**Break (Questionnaire)**
17.30 – 18.30	The assignment: Keep Alive Red Dot
18.30 – 19.00	Group Reflection (Interview)
19.00 – 19.10	Distribution of gift cards

Magnus Bärnholt, Andreas S. Lyngby

An Internet-of-Things Software Framework for Exergames

NTNU
Norwegian University of
Science and Technology