Oregon State University
College of Engineering

CS Capstone Design Document

November 24, 2019

DreamZBox 2.0

Prepared for

Oregon State University EECS

Megan McCormick
Corinna Brown

Prepared by

Group29
CS29 Dreamz Catcherz

Haofeng Tian
Cory Hayes
Maxwell Evdemon
Tristan Hilbert

Abstract

The stakeholders, Megan McCormick and Corinna Brown, have expressed concerns about the *DreamZBox 2.0*, a portable and cheap console. These concerns surround the entertainment, functionality, usability, and community for the system. The hardware interface, software composition, and game breakdown prove these concerns are met. First, the interface to hardware will provide functionality through abstraction in system services. Second, the dashboard software will ensure usability through segmented composition. Finally, the game's design and mechanics will adhere to entertainment and community concerns by focusing on the representations of player actions in-game. For packaging, the game will rest in the dashboard's software composition. Both software pieces will reside on the console's default *Linux* operating system. The team felt these design choices would best orchestrate the needs for data and processing within the limited capabilities of the *DreamZBox 2.0*. The CS29 Dreamz Catcherz will proceed with these designs at the discretion of the stakeholders, finishing the project by May at the latest.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

The *DreamZBox 2.0* is a video game console, made cheap and portable. The console will host the game, *Maze of Dreamz*, created by the *CS29 Dreamz Catcherz*. *Maze of Dreamz* represents a playable experience to express the capabilities of the *DreamZBox 2.0* console. The *DreamZBox 2.0* console will also host a dashboard called *DreamZDash*. These software solutions aim to create an enjoyable user experience for the *DreamZBox 2.0*. The following sections work to uniquely identify this document. The background, contexts, and concerns are provided. The formalities and terminologies are also defined in these sections. These details make the project and software more understandable to a larger audience. They aim to provide better understanding before detailing the designs in the body of the document. All design concerns must be established to fully portray the competency of the software system. These can be magnified by examining the scope provided in the next section.

2 SCOPE

This document will focus on design aspects of an atomic software system. Breaking down the solution requirements, the concerns are grouped and defined in the body of the document. Software design views aim to meet sets of concerns based on chosen design viewpoints[1]. The views explain which concerns the viewpoint meets. The viewpoint dictates which concerns the view can meet. It also demonstrates the necessary elements needed to implement its view. The elements resemble the required tools, technologies, and structures. All concerns are met by one or more design views within this document.

3 CONTEXT

The designs work to encapsulate the software for a specific video game console. The *DreamZBox 2.0* uses a proprietary controller with a customized interface. The innovative controller uses wireless hardware to transmit user input. Games on the *DreamZBox 2.0* will make use of all features on the custom controller. The *DreamZBox 2.0* will also come with a virtual hub (or dashboard). The dashboard will provide selections of settings and games for the user. Both the game and the dashboard adhere to the designs within this document.

4 DESIGN LANGUAGES

The document produces the following design languages to detail the different viewpoints of our document. These all provide different conceptualizations of system components.

4.1 UML Case Diagram

A UML Case diagram exposes the external inputs and outputs to a viewpoint. It provides potential external actors or systems outside the system boundary. The diagram details the required services which the system must provide for external actors. These services are also known as ”use cases.” The system or subsystem must contain each of these services. Some services have relationships with each other. One use case could help, extend, or cause another use case for a different actor. This language helps define the requirements of a system outside of its technological constraints. This leads to potential depictions of functional break-downs and data couplings.

4.2 State Transition Diagram

A State Transition Diagram details different necessary modes for a system. It classifies each mode then describes the transitions between them. Each state embodies certain objectives or services that must be performed until a transition occurs. This could organize the necessary objectives into possible applications or subsystems within a designed system.

4.3 UML Class Diagram

The UML Class Diagram is different than the UML Case Diagram. The UML class diagram provides structured formats for compositions of data. In other words, it denotes what each entity in the system must contain. Numbers, identifiers, and arrays, represent different data components which can attach to a given class. The classes can also expose relationships to each other. Some classes may contain other classes. Some classes may resemble other classes. These relationships simplify translation to code. This remains extremely true in object-oriented programming languages like C++.

4.4 Sequence Diagram

A Sequence Diagram provides a row of different systems or entities which transmit information to each other in a sequence. These are perfect for showing the transmission of data given an event or use case. The diagram exposes the relationships and dependencies between entities or systems. It also provides some functional decomposition and interface definition between coupled entities. Sequence diagrams, working together within a UML Interaction diagram, provide more definition to the system. For simplicity, a single sequence diagram is used to frame the general use-case.

4.5 UML Component Diagram

The UML Component Diagram provides an outlook on a given system's composition. It breaks down the system into pieces. These pieces could translate into processes, classes, or other programming modules. These show the modularity of the design. It also portrays dependency and common structures within the design. This way certain programs may be used elsewhere in the code. The component diagram does not provide strict data-types, thus allowing large monolithic concerns to simply break down into manageable systems. The UML Component Diagram acts as an excellent format to decompose a system into programmable modules[2].

5 VIEWS

The following views are given for the document.

- Hardware to Software Interface
- Dream Modes
- Dashboard Service
- Game Experience
- Game Logistics

6 SUMMARY

The design for the *DreamZBox 2.0* will focus mainly on points of interest. These are the case diagrams for the game, *Maze of Dreamz*, the dashboard, *DreamZDash*, and the hardware to software interface. Each of those three parts make up our goal of creating a unique game and user interface. The game's design focuses on the game itself and the actions the user can take within that game. The dashboard design deals with the start up of the console and the actions taken within the user interface. The hardware to software context design deals with inputs from the user, then the effects it has on the console and the software.

7 GLOSSARY

Bluetooth	A wireless connection interface using radio transmissions in hardware.
Disk	Slow long-term storage that maintains after the hardware powers down.
Driver (Software)	Software which provides decoding and use of serialized hardware data.
Hardware	The physical and electrical components of the system.
Interface	A layer of communication or transmission between the user and system.
Non-Volatile Memory	Data storage on a machine that retains integrity after the machine powers down.
Run-Time	The period where an application or piece of code is being used.
Software	The programmed components of the system. These components work upon hardware (see hardware).
Throbber	A graphical swirl or animation that denotes operations occurring in the background.

8 DESIGN STAKEHOLDERS

Corinna Brown and Megan McCormick act as the clients for the *DreamZBox 2.0* project. They worked on the original version of the DreamZBox. They want a prototype version of the *DreamZBox 2.0*. Alongside the *Electrical Engineering and Computer Science* (EECS) department of Oregon State University, they provide the funding for the project. They desire a functioning game and dashboard system on the perspective prototype system. The following requirements and concerns describe the aspects they seek.

9 CONCERNS

9.1 Fun Experience

The stakeholders want a pleasant experience within the *DreamZBox 2.0*. This includes a video during the launch of the console. Also, the console will provide an innovative game to spark curiosity for the console. *Maze of Dreamz* and *DreamZDash* must not hamper or create obstacles for the user. In all cases, the software should provide entertainment for the end-user.

9.2 Functional Interface

The software on *DreamZBox 2.0* must function adequately. In other words, the graphics load from memory instead of disk. Secondly, the console adequately decodes input. Thirdly, the *DreamZDash* should show hardware status (i.e. battery life). Finally, *DreamZDash* and *Maze of Dreamz* need to allow transition between each other. All of these features must be accessible to the end-user. These should be in the form of graphical user interfaces manipulated by the Bluetooth controllers.

9.3 Descriptive Interface

This concern coincides with functional interfaces, but the user interface needs to be descriptive. The graphics used within *DreamZDash* and *Maze of Dreamz* should always show what is happening in the system. The user should not need to question the current state of the system. Throbbers for loading and settings for graphics should allow for users to feel in control. The interface should include components like these to provide specificity within its design.

9.4 Multiplayer Game

The *DreamZBox 2.0* will be packaged with two controllers. This means the game needs to have a multiplayer aspect. The design should not assume a single playable entity. The camera, data, and mechanics in *Maze of Dreamz* should hold for both one and two users.

10 Gantt Chart

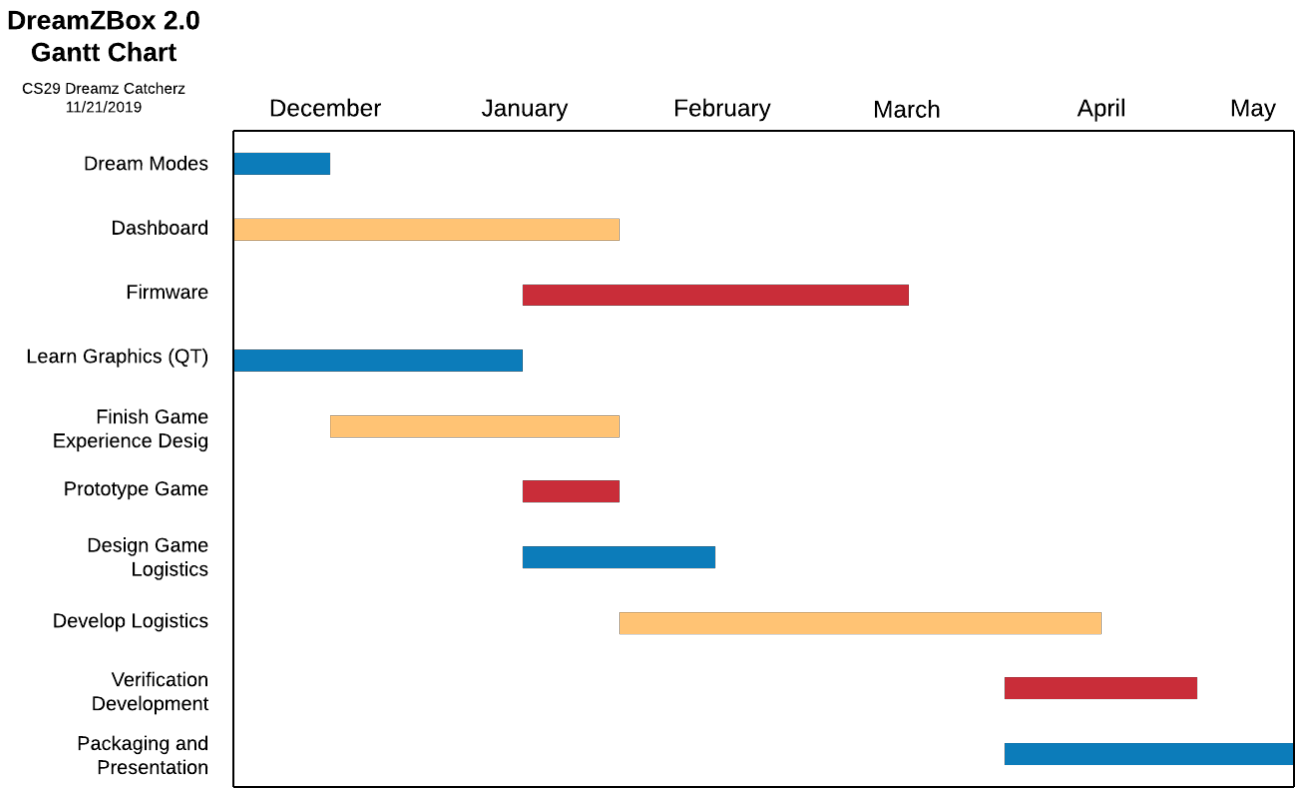

Fig. 1. The *DreamZBox 2.0* Gantt chart displays the plan for the next several months in implementing the viewpoints. The Gantt chart also includes other tasks which may take precedence during development. The Gantt Chart establishes the latest possible deadline for the given pieces of the project.

11 Hardware to Software Interface

This view considers the concerns towards device interaction of the *DreamZBox 2.0* with the external world. This mainly pertains to the *Functional Interface*. For this view, the best viewpoint would be a context viewpoint. This will abstract the system's services from the external forces, framing the concerns dependent on external actors. This will help define where the software drivers need to exist in the system.

11.1 Design Elements

Any user or player will be considered a *player* within the use case chart. The output for video and audio of the system will be considered the *television*. The external storage for the game will be an actor called *Game Store*. The battery will be considered a *Battery* actor. The system responds to the different operations which can occur by these actors. The use cases within the diagram define these responses. The software must administer these use cases.

11.2 Design Language – UML Case Diagram

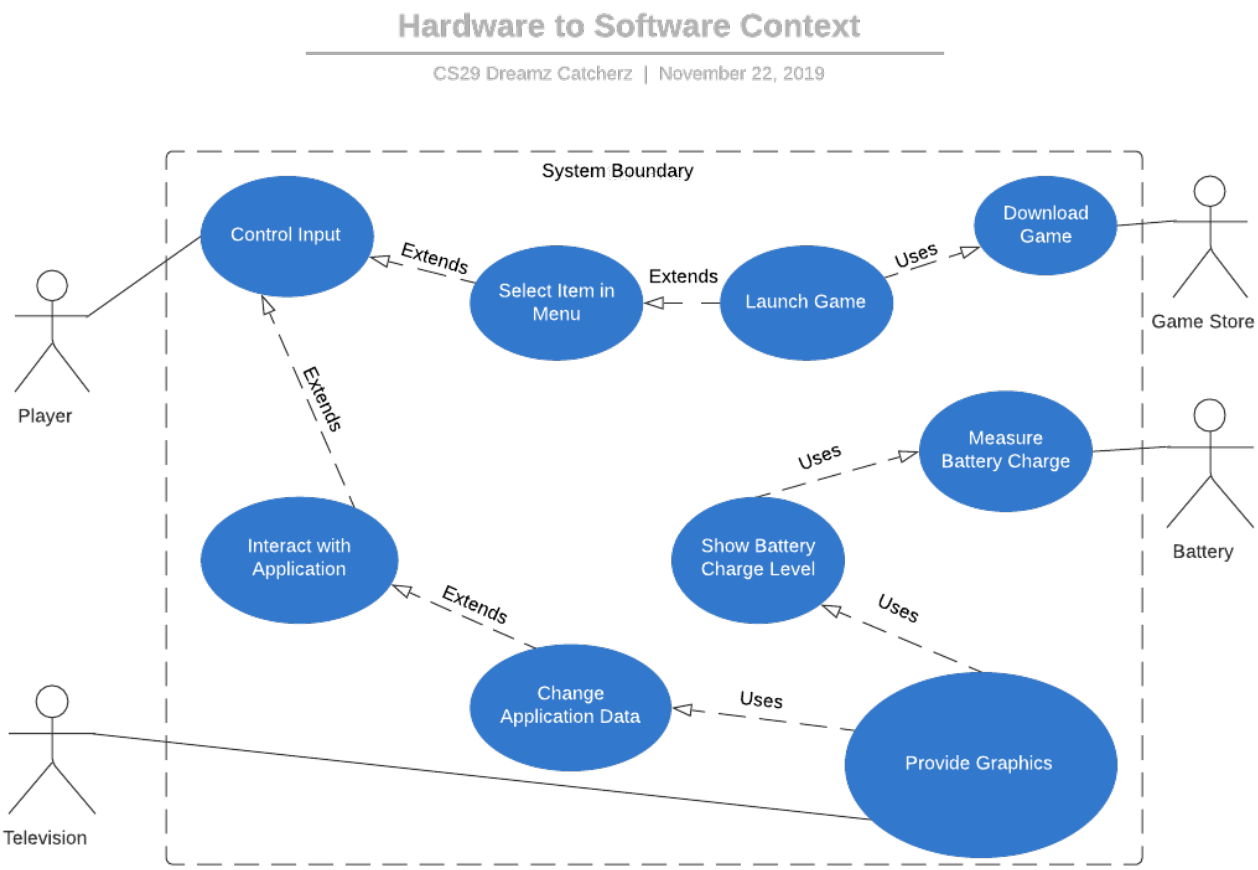

Fig. 2. The UML Case Diagram establishing the Inputs and Outputs of the system. It's important that the external actors of this diagram could be translated to components in the *DreamZBox 2.0* hardware. Also, the different arrows establish relationships between the provided services. The "extends" arrows provide an extension to a given service. The "uses" arrows determine services which use the output of another service.

11.3 Design Mentality

The design establishes the different services which must be provided to the external actors. The actors could resemble a hardware context. A *player* represents the Bluetooth input. The *television* represents the display port. The *Game Store* could represent the WiFi antenna. And, the *battery* could represent the battery sensor. Per context designs, the components do not matter. These actors were placed outside the boundary to define the inputs and outputs of the entire system.

This way an operating system like *Linux* could be used to interface to all of the external forces on the system. Then, the system could reside within *Linux*, to facilitate functionality for these use cases. The design also shows lack of coupling between actors' services, which allows for better performance within the design architecture. This format of design, allowing all services to act independently, will improve performance.

12 DREAM MODES

This view takes into consideration the Dashboard/User modes portion on the *DreamZDash*. This designs around the concerns for a *Fun Experience* and a *Functional Interface*. The *DreamZDash* needs to provide a video upon launch. The *DreamZDash* must also provide the ability to launch a game. This viewpoint provides interface between the software components of *Maze of Dreamz* and *DreamZDash* for the *DreamZBox 2.0*.

12.1 Design Elements

The *Video* state represents a played movie file. During this state, the user cannot input. The user must watch the video. The Dashboard also uses the self-titled *Dashboard* state. This inquires the settings to be changed from the user. The user can launch a game from the *Dashboard* state launching into an *In Game* state. These states comprise all possible states within the diagram.

12.2 Design Language – State Transition Diagram

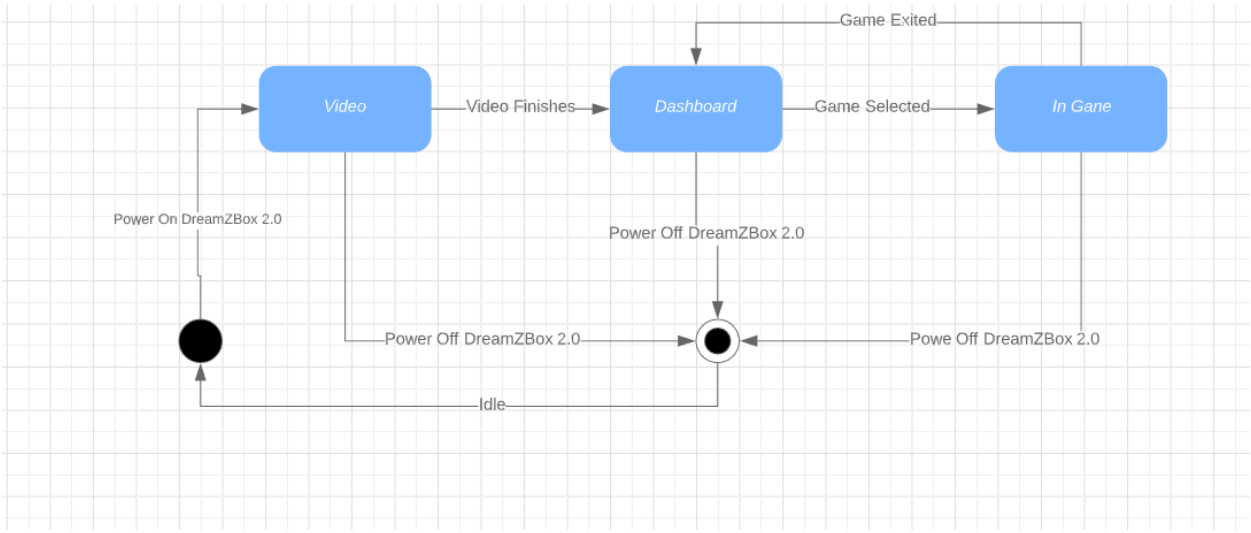

Fig. 3. This shows the different states the DreamZDash can enter into. As shown, the DreamZDash represents retains control of exit and entry into the system.

12.3 Design Analysis

The *DreamZDash* overlays the start up of the *DreamZBox 2.0* and the process the user goes through when selecting a game. It also dictates the transition into further states. This means the *DreamZDash* will have some control over the orchestration of the system. The design organizes the necessary outputs for each mode. It also dictates which domains each software portion covers. The *Video* state will not have control over the operating system settings like the *Dashboard* state. Overall, the design well defines the order and organization of the different software states.

13 DASHBOARD

This view depicts the necessary programming blocks to provide the *DreamZDash*. A dashboard within an operating system is extremely hard to describe. The user will need an understandable but abstracted view of the system. The user must experience graphics within specific contexts. The graphics could come from games, settings, or hardware in the console. The viewpoint governing this view must organize how it obtains and stores this data for graphics in a meaningful way. This view provides to the areas of a *Fun Experience*, *Descriptive Interface*, and a *Functional Interface*. By acknowledging how the data must be collected and maneuvered, the Dashboard view requires a composition viewpoint.

13.1 Design Elements

To allow for better translation into code, the orange outer components represent processes. These host a number of object instances, which represent the blue blocks. This excludes the *DreamZKernel* which acts as a singleton-class-process. Please refer to the design analysis in section 13.4 for further context. The relationships are also denoted between classes. The diagram's circular inputs meet to socket-looking outputs. The relationships show a general movement of data from one class to another. Overall, the UML Component Diagram provides an insight into a possible process breakdown during the run-time of the dashboard.

13.2 Design Language – UML Component Diagram

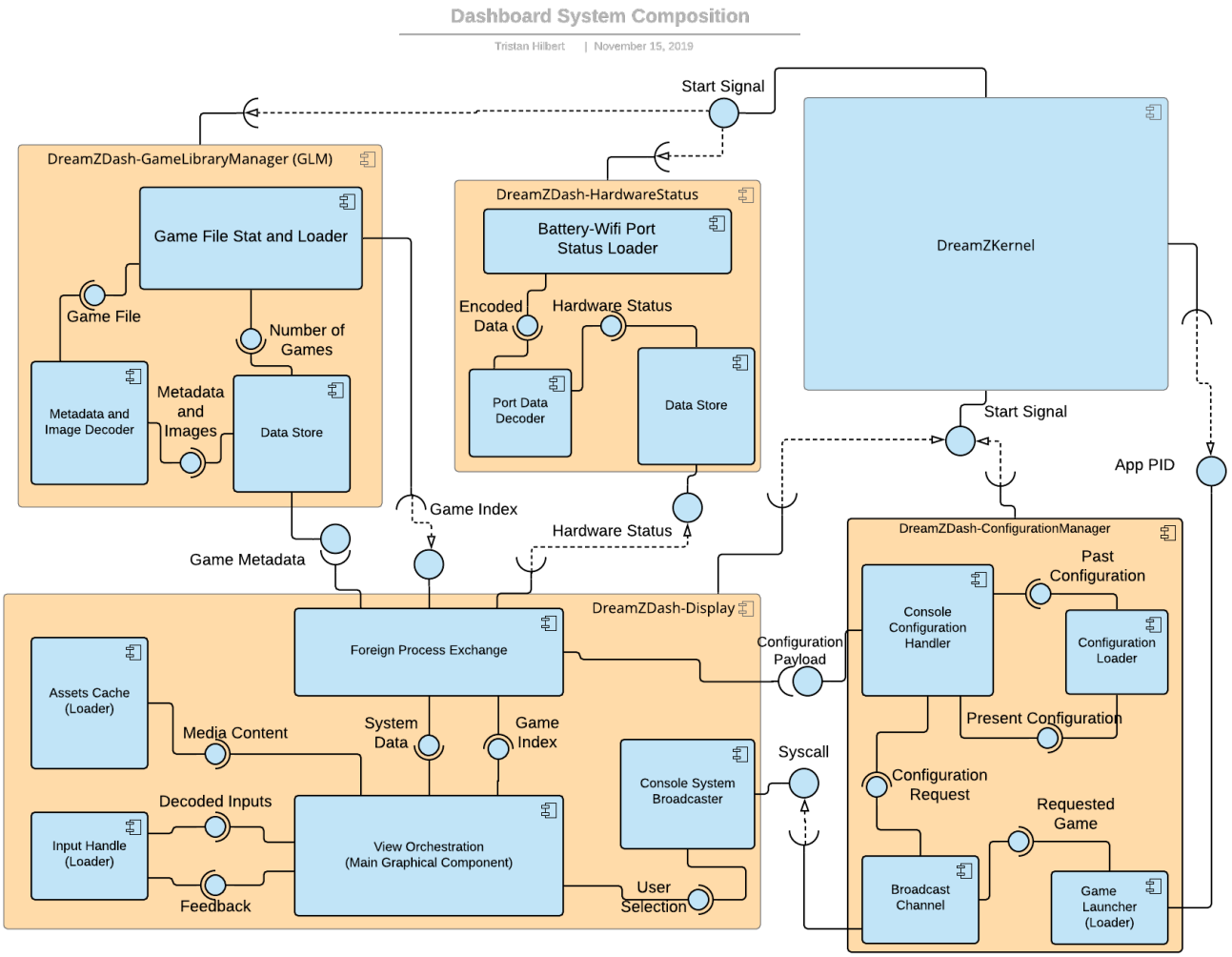

Fig. 4. The diagram above shows the different integral processes required for the dashboard system. The orange outer-components represent processes. The blue inner-components represent classes or objects. These build up the necessary data-flow for the dashboard system.

13.3 Design Language – Sequence Diagram

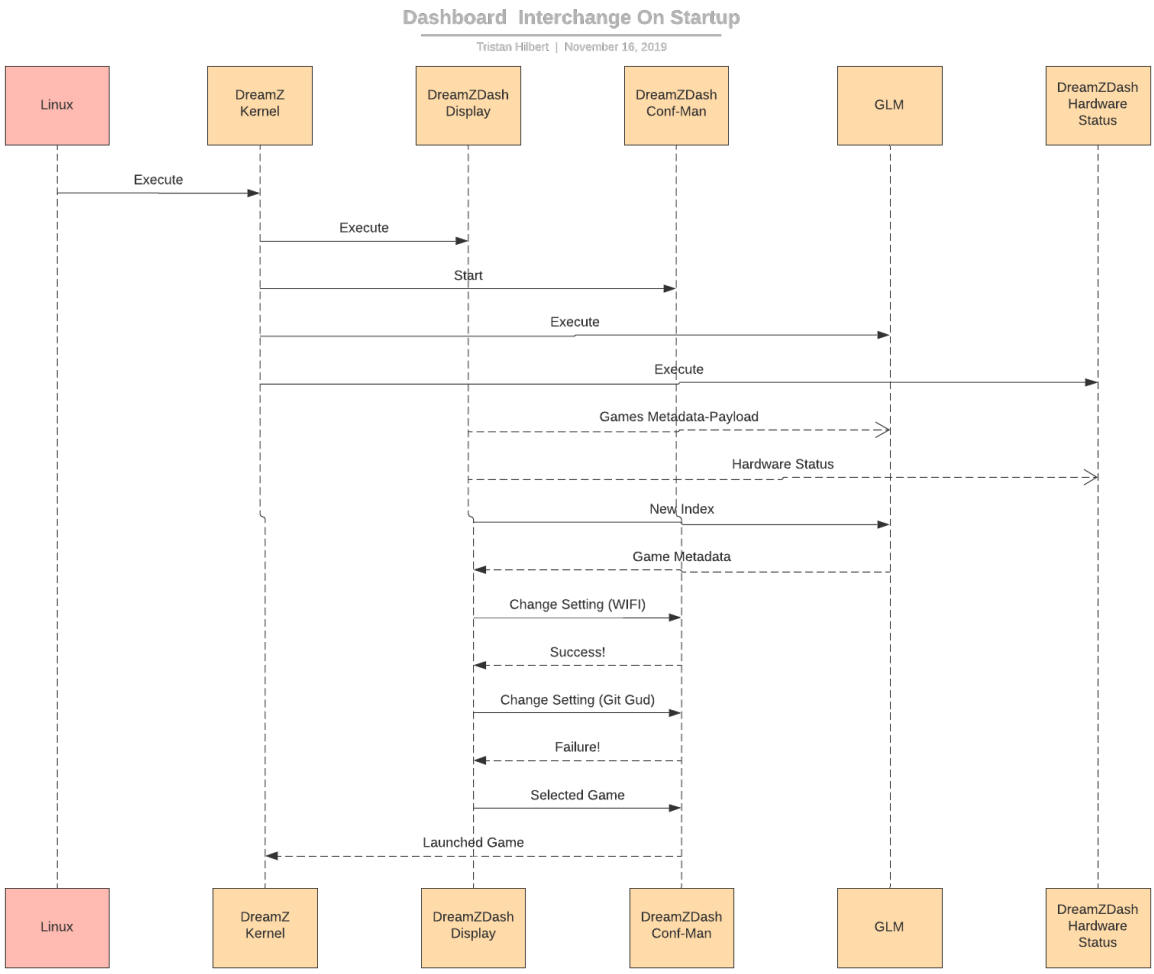

Fig. 5. The sequence diagram shows the order of calls and an example of interchange. In this example, the program starts up from scratch. The user then tries to change a few options before launching a game. Some of the options are respected, but the transmission occurs sequentially.

13.4 Design Analysis

The *DreamZKernel* represents the parent software of the entire *DreamZBox 2.0*. In other words, the *DreamZKernel* should always run on the system, orchestrating the running processes at all times. This means the process must require as little memory as possible. The *DreamZKernel* will also keep track of the *Dream Modes* mentioned above, but the *DreamZDash-Display* will be in charge of playing the video during asset load. The DreamZKernel will be in charge of launching other programs. The order of process launching should be the following:

1) *DreamZDash-Display*
2) *DreamZDash-ConfigurationManager*
3) *GLM*
4) *DreamZDash-HardwareStatus*

While the sequence diagram shows the fine order of these messages, the processes *should not* wait until a message is received. Each process should use their old data until replaced with new readings or returns. The sequence diagram only works to clarify the orchestration of potential functionality within the dashboard system. In summary, the diagrams provide the best general use for the envisioned dashboard. Some patterns within the composition diagram could use further definition. These patterns of components are defined below.

13.4.1 Loaders

Many of the files and data loaded during the consoles start-up will be organized before hand. This shows the benefit of using non-volatile memory. The ”loaders” will have a designated filename or address to always be reading or streaming data from. This means each one will have a dependency on the *Linux* operating system not shown in the diagram. It is recommended that these classes have a caching form or programmed container within their functionality. This will increase performance and speed since much of the graphical concerns require these data components.

13.4.2 Inter-process Exchange Coupling

In particular, the *Foreign Process Exchange* component depends on data from all other processes. This data coupling is abstracted, leaving some simplicity in implementation. The implementation of the *Foreign Process Exchange* object should be handled with care. With research, the recommendation from the designer is to use shared memory within the POSIX standard[3]. The Boost libraries within C++ provide the same ability[4]. The component will require extensive error handling. The implementation should always consider situations where *all other processes* do not exist. The *DreamZDash* should have an expected function without its dependencies met.

13.4.3 View Orchestration

Even after design, the *View Orchestration* component will need immense decomposition. The dependencies on graphical interfaces should be denoted later in design. This component is abstract in this diagram, because the chosen libraries like QT should provide all necessary dependencies. It should also be mentioned, that the Assets Cache and View Orchestration will heavily couple. The View Orchestration cannot display without the assets. This information is important for concerns within implementation, verification, and profiling. Should the View Orchestration object be broken down, perhaps the grouping of functions or component objects could allow a fast path for showing graphical media like the video.

14 GAME EXPERIENCE

The game experience derives majorly from the game’s core mechanics. The game mechanics provide the important architecture for a game’s design. While this document has intense focus on software, the mechanics of the game provide description on player experience. These respond to the concerns of *Fun Experience* and *Multiplayer Game*. By defining how the game should play, the software can follow the algorithmic procedure.

14.1 Design Elements

The different use cases within the diagram represent different actions for the player. These are game mechanics or functional descriptions necessary for the game. The mechanics describe the experience for the player. As they play through the game, the software must communicate these experiences through graphics and sound.

14.2 Design Language – Use Case Diagram

See section 18.1 for the Use Case Diagram.

14.3 Design Analysis

The *Maze of Dreamz* overlays the whole process of a player starting up *Maze of Dreamz*, playing it, and ending the game. At first, the player has to power up the *DreamZBox 2.0*, then select the *Maze of Dreamz* in the *DreamZDash* to start the game. During the game play, the player can interact with monsters, items and checkpoints. A battle event will occur when the player gets close to a monster. He can use the attack button to deal a small amount of damage immediately or use the skill button to cast some special skills after a certain time. The monster will automatically attack the player when he gets close enough. Items can be picked up on the map and be consumed in the bag. When the player has no less than 1 key in the bag, he can use the key to pass the locked door. The checkpoints can be set by the save button. The player can teleport to the newest check point anytime he wants using the teleport button. When the player loses all hit points, he can also revive at the nearest checkpoint.

15 Game Logistics

Data architecture remains integral to the project. The game requires organization of the game's various data fields. These will conform to objects. This view provides to the *Functional Interface* and the *Multiplayer Game* concerns. This viewpoint respects the requirements for encoding the game into software. The necessary containers for data need to be defined for the best implementation.

15.1 Design Elements

The design elements represent the class hierarchy the game programming will follow. C++ will be used, so the different classes translate straight into the language. The diagram also provides ”extends” relationships which mean the origin of the arrow uses all of the fields at the destination. The ”relation” arrow describes an class being used within another class. In other words, one of the fields in the destination of the arrow is defined by the origin of the arrow. These relationships and diagram reveal the necessary objects for the designed game.

15.2 Design Language – UML Class Diagram

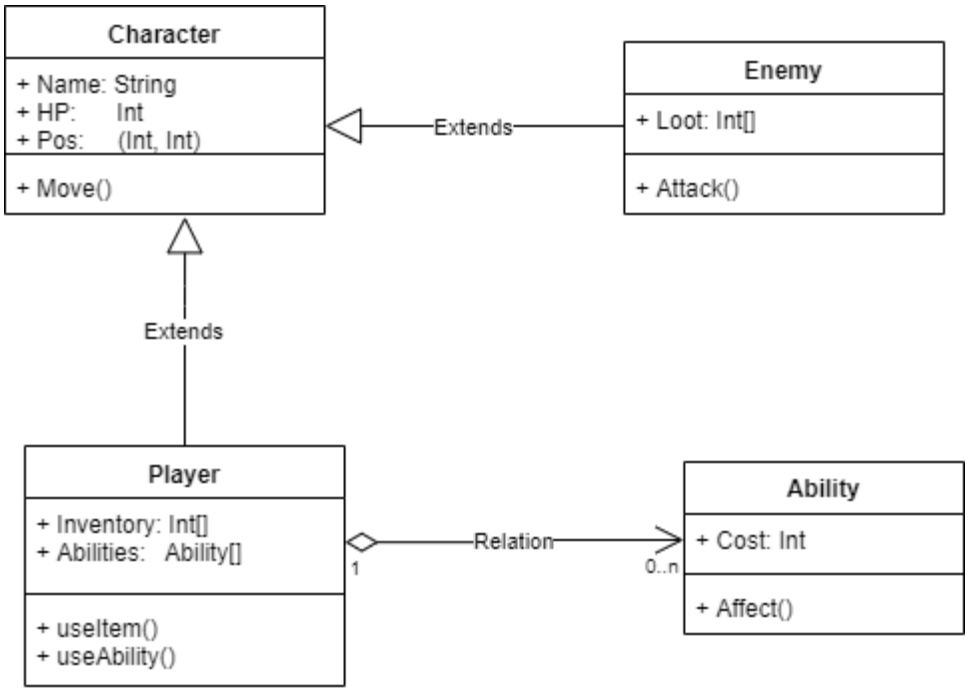

Fig. 6. This UML class diagram depicts the necessary data-types for the game's mechanics. The character provides an ample class used by all "living" entities. This means characters that have a "living" state and a "death" state. The player has these and a set of abilities denoted by it's ability list field.

15.3 Design Analysis

The character seems to represent a "living" datatype which could be used in a majority of the program. This will classify really any object instance which can move. The is shared with the *Pos* or position field depicting a tuple for two dimensional position. This could also be considered a coordinate on a two dimensional plane. The player will then have functions we could attach to input interrupts with an events module. Overall, the classes listed in the diagram provide necessary implementation for the game.

16 Design Rationale

16.1 Viewpoint Interaction and Rationale

For the *DreamZBox 2.0* the viewpoints interaction is as follows. The *Hardware to Software* deals with player interaction. That player interaction is then used to decide the state of the dashboard, which can be seen in the *Dream Modes* viewpoint. These transitions control how the user changes the state of the *DreamZBox 2.0* and the *DreamZDash*. On a deeper level, the viewpoint for the states of the *DreamZBox 2.0* interacts with the *Dashboard* component viewpoint. In which the viewpoint makes sure to load and display the correct data based on the current state of the *DreamZBox 2.0*. When the state of the *DreamZBox 2.0* proceeds into the *Maze of Dreamz*, the component viewpoint directs to both the *Game Experience* viewpoint and the *Game Logistics* viewpoint. This component viewpoint for the *Dashboard* suggests how to load the correct stored data based off the state of the user in game. The *Game Logistics* then provide the programming translation to the game mechanics within the *Game Experience* viewpoint. The rationale behind these viewpoint interactions is to make sure the

possible options of the player and the code line up with one another. In this case, the *Hardware to Software* components deal with player interactions which can change the state of the *DreamZBox 2.0*, thus changing what needs to be loaded for the component viewpoint. Then, the loading will include the data for the *Game Logistics* which embody the *Game Experience*. In that sense all the viewpoints are connected, each representing a different section of the *DreamZBox 2.0*'s functionality and building upon one another.

16.2 Viewpoint Component Diagram

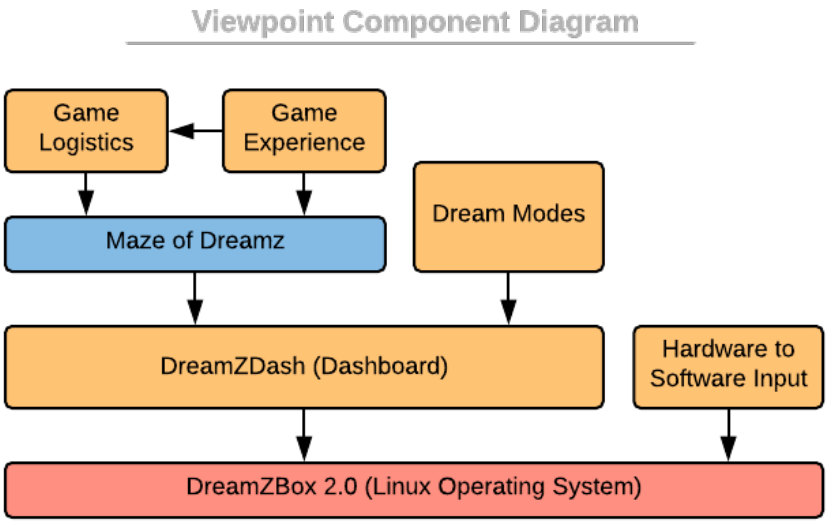

Fig. 7. The Viewpoint Component Diagram displays how each component produces into another viewpoint. This ends with the base-plate *DreamZBox 2.0* main operating system or *Linux*. All viewpoints presented will build upon this system. Each viewpoint possibly provides to one or more other viewpoints. Viewpoints pointing to the same viewpoint may also interact with each other.

16.3 Design Reflection

The necessary concerns for *DreamZDash* and *Maze of Dreamz* requires orchestration of process and data. These design choices define how the code needs to accomplish the concerns. Future designs outside this document will provide more detail in how each function will process. The processing of data remained the largest question during this design. The designs needed to plan the organization, extraction, and use of all kinds data. The team also wanted to view this project from a variety of angles, allowing for multiple types of viewpoints to be taken. The designs answer questions lacked in research. In conclusion, these designs work to define the necessary components for the many required tasks on the *DreamZBox 2.0* running the *Linux* operating system.

17 Conclusion

This document covers the different parts of the *DreamZBox 2.0* that our group is in charge of. This will mainly include the software sections of the *DreamZBox 2.0* along with some of the firmware connectivity. This document specifically deals with UML figures and dictated overviews of each part. This includes the User interface dashboard, the firmware and software connection context, and design for the *Maze of Dreams*. We have explained the requirements for each of these sections along with analysis of these parts. From this, we can start to draw conclusions on our own design ideas, all of which will be used when succeeding in our groups part of the *DreamZBox 2.0* project.

18 APPENDIX

18.1 Maze of Dreamz Flow Chart

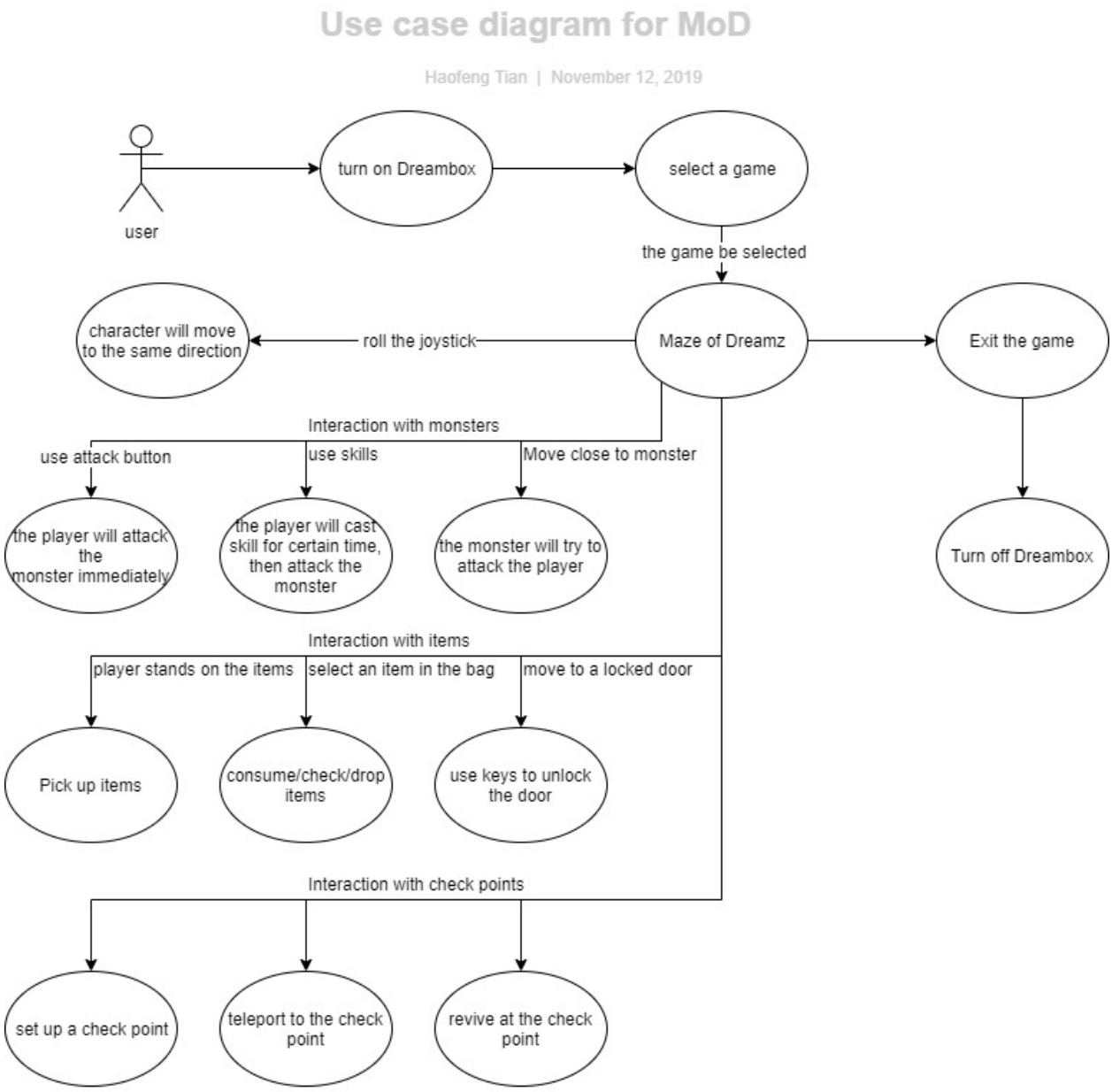

Figure 2: use case diagram with Maze of dreams

Fig. 8. The UML Case Diagram for all actions a player can take in Maze of Dreamz during the game play. The player should be able to have interactions with monsters, items and check points as it is listed in the diagram.

REFERENCES

[1] Institute of Electrical and Electronics Engineers, "IEEE Standard for Information Technology—Systems Design—Software Design Descriptions (IEEE Std 1016)," 2009.

[2] "UML Component Diagrams," 2010. [Online]. Available: https://www.uml-diagrams.org/component-diagrams.html

[3] Kjohri, "Interprocess Communication Using POSIX Shared Memory In Linux," 2017. [Online]. Available: https://www.softprayog.in/programming/interprocess-communication-using-posix-shared-memory-in-linux

[4] B. Schäling, "Chapter 33. Boost.Interprocess," 2008. [Online]. Available: https://theboostcpplibraries.com/boost.interprocess