Bilkent University

Department of Computer Engineering

# Senior Design Project

*Coda*

## High Level Design Report

Merve Kılıçarslan, Yağız Efe Mertol, Ege Özcan, Çağla Sözen, Murat Tüver

**Supervisor:** Prof. Dr. Uğur Güdükbay
**Jury Members:** Prof. Dr. Halil Altay Güvenir, Prof. Dr. Fazlı Can

**Innovation Expert:** Prof. Dr. Veysi İşler (SimSoft)

High Level Design Report
December 30, 2019

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491.

## Contents

Project High Level Design

*Coda*

## 1 Introduction

Music as a form of art has been the common interest of people used to express feelings and identity through a composition of rhythm, timbre and melody. In different forms and sounds by making use of the variety of instruments, music is present in almost every context for numerous purposes. Besides the pleasure of listening, it’s been proven that playing instruments have positive effects on brain development, especially for spotting statistical patterns enabling the learner to better predict what would happen next in a pattern, so every child has the right to better themselves with the help of instruments insomuch as discovering their musical talents [1].

Over the last 20 years, the number of children learning to play an instrument or playing an instrument has increased significantly [2]. However 26% of children and 49% of adults in the UK stated that they’ve given up playing instruments although they’ve learned to play or started to [2]. Most common reasons for this are loss of interest, instrument cost, lesson costs and competing pressures from school [2]. Furthermore, the fact that some instruments are highly immobile by nature makes practicing very challenging for both individuals and for groups of people who practice together. In most cases, instruments become idle and forgotten because of the impracticalities mentioned. As a result, buying instruments may be seen as an unnecessary expense. When the cost of learning instruments and the instruments itself is taken into consideration in addition to immobility, it can be stated that instruments can be made further accessible.

Therefore, there is need for solutions to make playing and learning instruments more sustainable by making them more accessible in several aspects like cost and mobility and we believe that with the increasing number of smartphones, mobile phones can be used to address this problem.

## 1.1 Purpose of the System

To address the need for solutions to make playing and learning instruments more sustainable by making them more accessible in several aspects like cost and mobility, *Coda* will be an application that will use Virtual Reality and Computer Vision to simulate the experience of playing an instrument  for a user, enabling them to practice without workstation, cost and even sound limitations. By using novel Computer Vision techniques, we will eliminate the need for any hardware or tools and make the whole experience only depending on the smartphone, an AR cardboard and the gestures of the user. Different from the current products in the same context, we will provide a more realistic and immersive experience using Augmented Reality both visually and aurally.

## 1.2 Design Goals

### 1.2.1 Usability

Usability is crucial for *Coda*. It is the prior design concern during system design. Fundamental idea of Coda is to mimic the experience of playing an instrument, hence the success of the system is determined by the ability to mimic such real-life, tangible experience.

Therefore, *Coda* aims the following during system design in terms of *Usability*,

- The application should be user-friendly and as easy to use as a real instrument.
- Controls in the system should be straightforward and the instructions should be clear. The user should be able to choose a piece from the library and start playing in no longer than 1 minute.
- The application should have clear and concise instructions for initialization. The initialization stage and initial hand recognition should not take longer than 2 minutes when application performance is neglected.

### 1.2.2 Performance

Performance is a substantial requirement for *Coda*. Since once again, the aim is to mimic the experience of playing an instrument as intuitively as possible, hence the ability to provide a real-time performance with instant auditory and visual feedback is another essential goal. For this, we have made the design as efficient as possible in terms of functions and will utilise a cloud server if needed to provide better performance.

Therefore, *Coda* aims the following during system design in terms of *Performance*,

- Frames Per Second (FPS) should not drop below 25.
- Input lag(time between user gesture and recognition in the app) should be less than 2ms

### 1.2.3 Extendibility

The long-term goal of *Coda* is to offer users as many instruments as possible for serving our fundamental mission of accessibility. To address this goal, the design is made in such a way that the system is divided into proper sub-systems with highly cohesive and low coupled classes/packages.

Therefore, *Coda* aims the following during system design in terms of *Extendibility*,

- Design of the application code should be written in such a way that will enable addition of new instruments. For instance, all specific instrument classes will be children of the general *Instrument* class for extendibility purposes.
- Storage in the system should be designed in such a way that it can be extended to be shared over a network or by utilising the cloud.
- Design of the network usage should be in a way that it supports a potential implementation of **band mode** which allows people from different devices to collaborate and play different instruments at the same time.

### 1.2.4 Security

Since *Coda* will require access to the camera of the mobile phone of the user and record the footage for some features, it is important that the design of the system will take the security of these footages into consideration during design.

Therefore, *Coda* aims the following during system design in terms of *Security*,

- User data will not be shared with any third parties under any circumstances.
- No user data will be saved without the consent of the user.

## 1.3 Definitions, acronyms, and abbreviations

**UI:** User Interface

**FOV:** Field of View

**6DOF:** Six Degrees of Freedom

**Client:** Part of the system that the user interacts with.
**Server:** Part of the system that handles the access to a centralized resource or service.
**AR :** Augmented Reality. A computer enhanced version of the real-world where virtual objects can be put on real world objects in an interactive environment.
**VR :** Virtual Reality. A computer generated world that is completely virtual in terms of environment and the objects in it which provides a virtual interactive environment to the user.
**Computer Vision:** Combining/Using advanced Image Processing, Machine Learning and Deep Learning techniques to enable computers to see as a human does.

## 1.4 Overview

*Coda* will be an Android mobile application designed to make instruments more accessible in terms of cost and mobility. *Coda* will depend on a system that renders an instrument in an augmented environment on the smartphone's screen and creates an AR environment with the use of only a Google Cardboard like headset. Also, *Coda* will track the hand gestures of the user using the smartphone's camera by Computer Vision techniques in accordance with the FOV of the user.

As the main feature, the system will give visual and auditory feedback to the user according to the user interaction by gestures for simulating the experience of playing an instrument only keeping out the tactile experience. Additionally, the user will be able to choose between several modes: Free Playing Mode and Practice Mode. These features will enable the user to either practice freely or practice on a particular piece of their choice from the library. In the practice mode, the system will provide visual directives for the user to play the piece correctly, further enhancing the experience and providing ease for learning pieces. These will be able to be recorded and saved for enabling composing a new piece. For initializing the system, the user will have to align their hands and environment. The instrument will be placed according to this alignment and it's location will not be changed after initialization.

This report explains the details of the proposed system design in terms of Software Architecture including *Subsystem Decomposition, Hardware and Software Mapping, Data Management, Access Control Policies, Security, Global Control Flow,* handling of *Boundary Conditions* and *Subsystem Services* with the help of Unified Modelling Language (UML) for proper documentation of the system and clarity of the system design. We aim to come up with a System Design that complies completely with our Design Goals explained above and therefore our requirements as explained in the Analysis Report to construct a system that fulfills the need of our users while providing a usable and efficient application.

## 2 Current software architecture

There doesn’t exist a current system that the application will be built upon, but there are several similar products that tried to implement the same idea but with different fundamental objectives and therefore different implementations. For instance the major difference of *Coda* from the similar products will be the elimination of the need for controllers that are extra hardware which cause the product to be more expensive and less accessible. Although the products may seem similar in terms of idea, elimination of the controllers create a fundamental difference which distinguish the required software architecture from the current software architectures strictly. Hence, software architecture of *Coda* will be designed from scratch according to the requirements and design goals explained.

# 3 Proposed software architecture

## 3.1 Overview

As briefly discussed in the former parts of the report, *Coda* aims to provide a minimal architecture in terms of hardware. Therefore, the software architecture of the system is limited to the smartphone itself. In this section of the report, the planned architecture will be described in detail by presenting the Subsystem Decomposition and further explaining the layers described in the Subsystem Decomposition individually. Then mapping of the hardware and the software in the system is explained. Furthermore, management of the persistent data, access and security conditions and boundary conditions are discussed.

Further information and all documentation on *Coda* will be published on,

https://ege0zcan.github.io/coda/

## 3.2 Subsystem decomposition

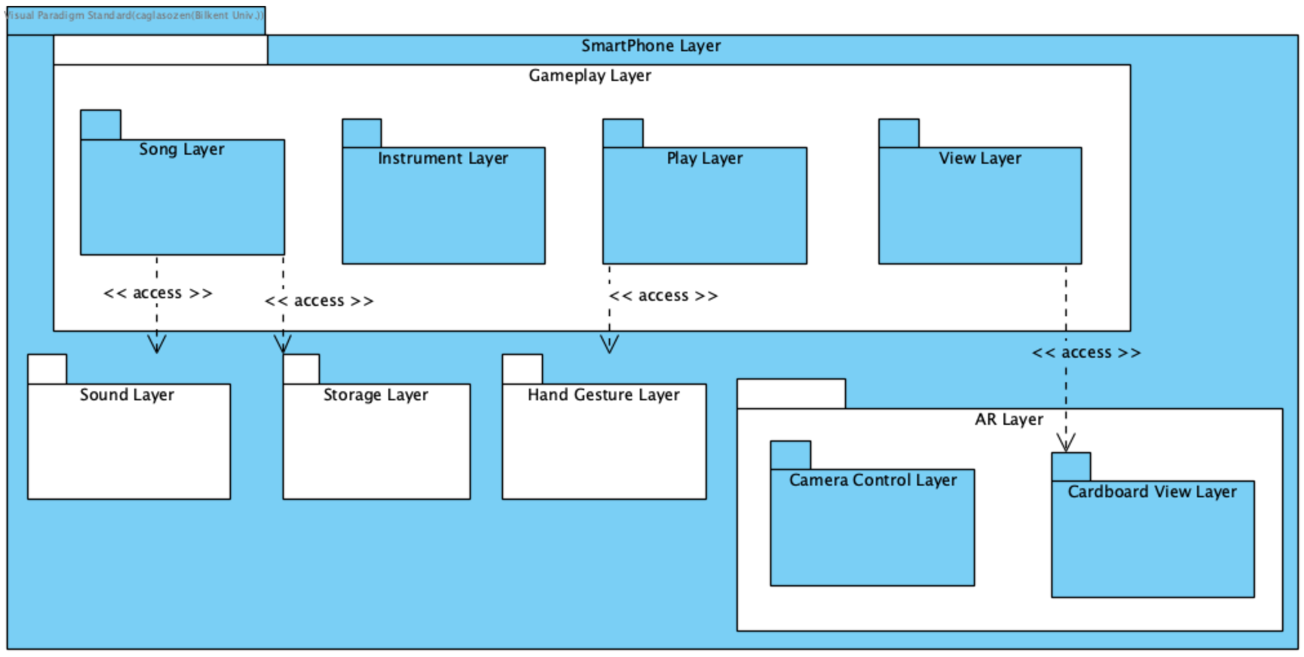

**Figure 1:** Subsystem Decomposition Diagram of *Coda*

*Coda*’s architecture is limited to the smartphone itself. All sublayers are within the *SmartPhone Layer*, which is the most comprehensive layer of the system. In this architecture, there is no distinction between the client side and the server side since both sides depend on the local system only. However this architecture may be subject to change if there occurs issues about the processing power of the smartphone.

Within the *SmartPhone Layer*, there are 5 sub-layers. The *GamePlay Layer* is the layer that provides the gameplay experience to the user, responsible from handling the relations between other layers and the client-side, in a sense. It accesses all other layers for synchronizing the AR, Sound, Storage and Hand Gesture components of the application with the user interactions. The *AR Layer* is responsible from handling the AR operations within the system and it is accessed by the *GamePlay Layer* for constructing the AR environment to be presented to the user as an augmented replicate of the real environment. For achieving this, the *AR Layer* also handles the information gathered from the camera of the smartphone. The *Sound Layer* is responsible from providing the auditory experience to the user, also interacting with the device speakers. This layer is also accessed by the *GamePlay Layer* for giving the the auditory response according to the user interaction. Storage Layer is the layer responsible from handling the persistent data management as explained in *Section 3.4*. This layer is also accessed by the *GamePlay Layer* for storing the library songs as well as the user recordings. Finally, the *Hand Gesture Layer* is the layer for handling and processing the *Hand Gestures* of the user from the camera and recognizing the user interaction, and intention, accordingly.
Further explanation of the above mentioned layers and their sub-layers is provided in *Section 4.0*.

### 3.3 Hardware/software mapping

Coda is an app that is designed to run on Android smartphones implemented on software and also using hardware components such as the camera and the screen. Although Coda is an AR based application, it requires no external hardware mapping other than the smartphone itself. Only the manual and straightforward integration of the smartphone to the Google Cardboard (or alike) headset is required.

Software components of *Coda* will be as follows,

- Java will be used as the primary software development language along with Android Studio for platform-independency, object-oriented tools and compatibility with useful frameworks and libraries such as Google MediaPipe, Google ARCore etc.
- Unity will be the platform for developing the graphics components.

All of the hardware mappings will be done by built-in Android SDK mappings to the smartphones running on Android. A simple representation of the Hardware/Software Mapping can be found below in the Deployment Diagram.

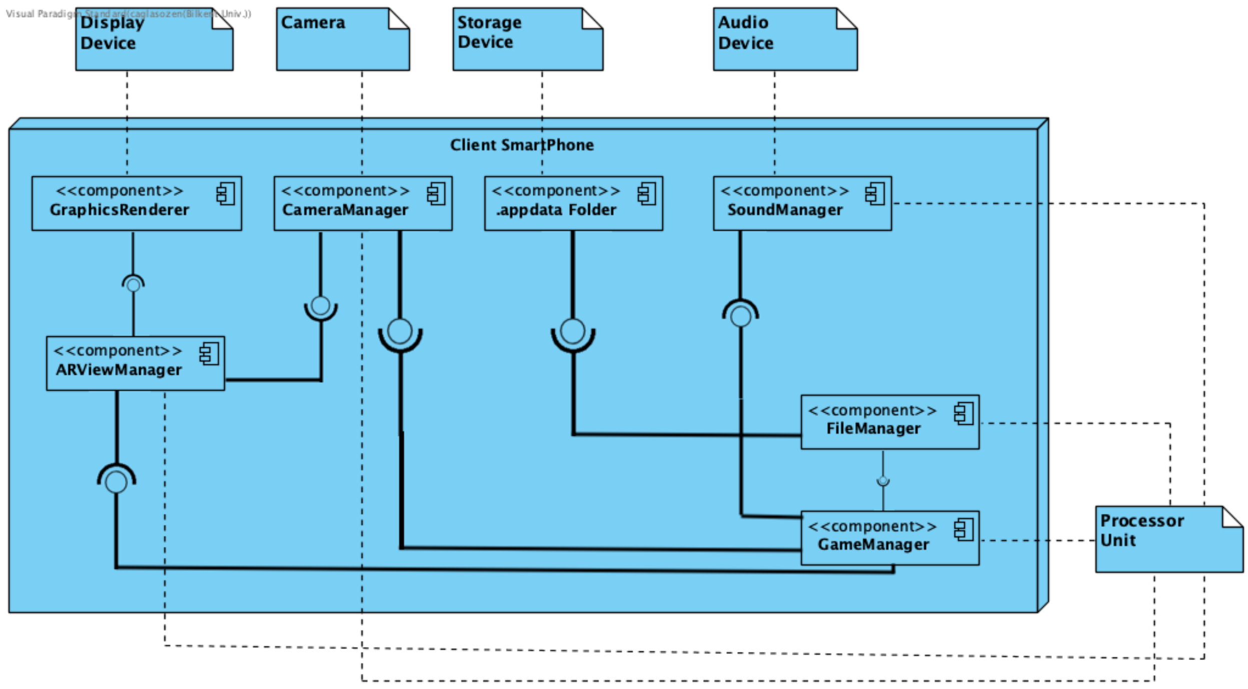
Display Device
Camera
Storage Device
Audio Device
Client SmartPhone
<<component>> GraphicsRenderer
<<component>> CameraManager
<<component>> .appdata Folder
<<component>> SoundManager
<<component>> ARViewManager
<<component>> FileManager
<<component>> GameManager
Processor Unit

**Figure 2:** Deployment Diagram of *Coda*

## 3.4 Persistent data management

Coda will require access to the persistent data storage of the smartphone to keep the information on recordings, library songs and user settings. Since it is required to recalibrate the hands and the environment for each use, session specific data is not a matter of persistent data management.

This data will be stored in a hidden folder named `.appdata` with several subfolders for different types of application data as follows.

- Library Song Data

  `/lib_songs.dat`

- Recordings Data

  `/recordings.dat`

- Settings Data

```
/settings.dat
```

For storage, it is not required to use an external database since the data to be stored is relatively small.

Approximate calculation of storage required for Coda is as the following. Note that the number of recordings allowed and library songs may be subject to change during implementation.

------------------------------------------------------------------------------------------------------------

| | |
|---|---|
| Approximate storage used for a single library song data: | 3-4 MB [3] |
| Approximate number of library songs: | 10 |
| Maximum number of recordings allowed : | 10 |
| Storage used for a single recording: | ~2.4 MB/minute [4] |
| Maximum length of a recording allowed (in minutes) : | 10 |
| The data to store the settings: | 1 KB |

------------------------------------------------------------------------------------------------------------

**Total:** [(10 x 3) + (10 x 10 x 2.4) + 0.001 ] ~= 270 MB

### 3.5 Access control and security

As mentioned in *Section 3.4* we will not use any external databases, only the internal storage will be utilised, so it is not required to implement an explicit access control mechanism. Furthermore, since the current version of *Coda* will not support network connection it will be unthreatened from malicious software originating from networks.

Once again, the discussed version of *Coda* will only support single player use which does not require any kind of sign-up procedure treating all users as a single type of user with the same access rights. This approach also ensures the security of the user data, which is in fact in minimal within *Coda*.

### 3.6 Global software control

In *Coda*, synchronization between layers are crucial for the user-experience, as the system is very interactive and dynamic in the sense that all responses depend on the user interaction whose source will be the device camera and will be recognized by the *HandGesture* Layer. Accordingly, auditory response will be given to the user which also requires a real-time synchronization. Furthermore, the sounds and the AR view also has to synchronized for showing the user the correct notes to hit on the instruments on time.

As explained above, synchronization is required among almost all layers as a result of the dynamic nature of *Coda*. For ensuring such synchronization, all layers that need synchronization are connected to the *GamePlay Layer* which will handle the synchronization among all connected layers. Namely, the *GamePlay Layer* will handle the synchronization of the following layers with the user-side,

- Input recognized by the *Hand Gesture Layer* will be handled by the *GamePlay Layer* to help the user navigate in *Coda* and interact with the AR environment by the *Play Layer.*

- Camera view received from the *Camera Control Layer* will be handled by the *GamePlay Layer* to construct the interactable user view by the *View Layer.*

- Songs are stored in the Storage Layer and it is accessed by the GamePlay layer to handle loading and playing songs in *Coda* in a synchronized manner with the view by also accessing the *Sound Layer*.

Since there exists no user specific operations in the current version of *Coda*, no synchronization is needed for concurrent accesses to a shared source.

## 3.7 Boundary conditions

### 3.7.1 Application setup

After installation of the app from the provided platforms (i.e. Google Play Store), Coda will require access to the device camera. Once the user allows access to these components, *Coda* will be ready for use. For startup, files containing library songs and recordings will be read from `.appdata` hidden folder. Coda will start up with the main menu screen for further routing into the application according to the user choices.

### 3.7.2 Session Initialization & Calibration

For starting a session, regardless of mode selection, the primary requirement for session initialization will be hand and environment calibration. This process is a prerequisite for the further parts of the application. In other words, the application can not be used without completing this step.

### 3.7.3 Recalibration

At any point of any session, in case there is a problem with the current calibration the user will be able to proceed to the calibration screen once again to correct the calibration. However in such case, all session data, i.e. progress, will be lost.

### 3.7.4 Termination

#### 3.7.4.1 Application Termination

*Coda* can be terminated like regular applications by exiting the application. With this feature, the user will be able to exit the application regardless of the screen they are in. However in such case, all session data, i.e. progress, will be lost. This applies for the case of an application pause since it is very likely that pausing the application might require recalibration.

#### 3.7.4.2 Session Termination

A session of *Coda* can be terminated from the *Exit* button in the in game menu. In such case, if there is a recording process, it will be automatically saved and added to the recordings library.

### 3.7.5 I/O Exceptions

Since *Coda* utilizes the device camera, speakers and the local file system, I/O failures are possible. Possible I/O exceptions can be listed as the following,

- **File Reading/Writing:** If there occurs a problem during reading songs from the library and writing recorded sessions to the local file system, *Coda* may have issues proceeding to the relevant screens. This also applies for showing the list of songs in the library and the list of recordings.
- **Corrupted Data:** If there occurs a problem with access permission issues for the local file system, saving recordings may be problematic. In such case, this problem

may be resolved by updating the access permission settings for the related folder through giving necessary instructions to the user.

- **Device I/O:** If there occurs a problem with the access permission settings of the device camera or device speakers, then *Coda* might have issues displaying the AR representation of the environment and giving auditory feedback according to user interaction.

### 3.7.6 Critical Errors

If there occurs a critical error causing an application collapse, *Coda* will recover in the same manner as a regular startup. However in such case, all session data, i.e. progress, will be lost.

## 4 Subsystem Services

### 4.1 GamePlay Layer

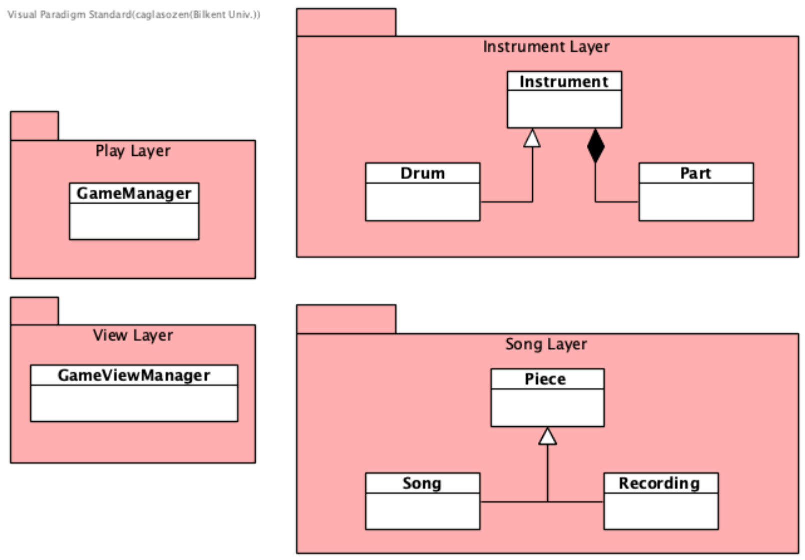

**Figure 3:** Subsystem Decomposition for the GamePlay Layer of *Coda*

This layer is treated like the client-side layer of *Coda*. Sub-layers of this layer handle the control flow with the device units like storage, camera and audio. Furthermore, this layer handles the user interaction and stimulates other layers accordingly. Sub-layers of this layer is designed in such a way that synchronization of all external layers will be handled by at least one sub-layer of this layer. This way, the user interaction and response to the user will be synchronized with the external devices and data.

Functionalities and structures of these sublayers will be further discussed in the proceeding sub-sections.

### 4.1.1 Song Layer

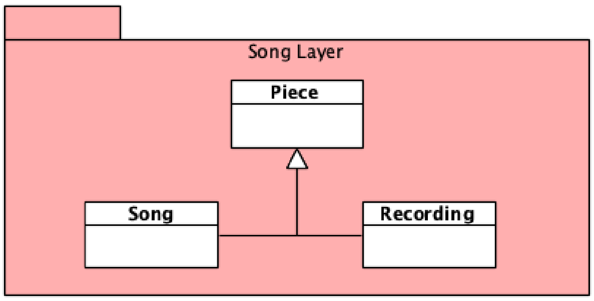

**Figure 4:** Subsystem Decomposition for the Song Layer of *Coda*

This layer is for handling the operations on the songs in *Coda*. Since this is a class within the *GamePlay Layer,* this class layer will contain the structure and definitions of playable structures of *Coda* like songs and user recordings. Both of these structures are defined as *Piece*’s in *Coda,* which represent the playable structures. This layer will handle fetching of stored songs from the local memory, and playing the pieces during run time in sync with the other layers.

#### 4.1.1.1 Piece Class

This class represents all playable structures in *Coda*. It is a parent class of the Song and *Recording* classes. It contains the mutual definitions of these playable structures in a general manner and invoke the functions according to the shared behaviour of the child classes like being played or fetched from memory and so forth.

#### 4.1.1.2 Song Class

This class is a specialised version of *Piece*. This class is a representation of the library songs. The distinguishing feature between *Song* and *Recording* classes is that pieces represented by the *Song* class need to be preloaded by the application developers in the current version of *Coda.* Songs are fetched from local memory and played by interacting with the *Sound Layer*.

#### 4.1.1.3 Recording Class

This class is a specialised version of *Piece*. This class is a representation of the user recorded songs. The distinguishing feature between *Song* and *Recording* classes is that pieces represented by the *Recording* class need to be recorded by the user and saved.

Recordings are saved into local memory, fetched from local memory when needed and played by interacting with the *Sound Layer*.

### 4.1.2 Instrument Layer

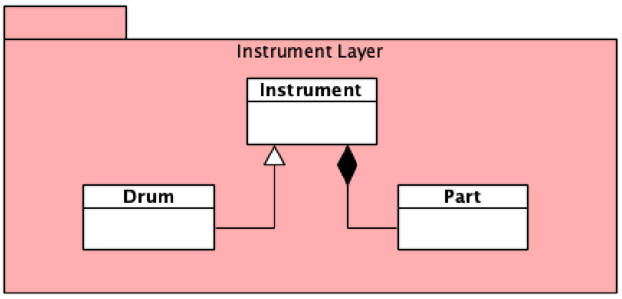

**Figure 5:** Subsystem Decomposition for the Instrument Layer of *Coda*

This layer is for handling the operations on the instruments in *Coda*. Since this is a class within the *GamePlay Layer,* this class layer will contain the structure and definitions of interactable structures of *Coda* that are virtual representations of instruments and their parts in AR. This layer will handle the interactions of the user with the defined instrument structures and the parts that the instruments are composed of, depending on the instrument structure.

#### 4.1.2.1 Instrument Class

This class represents all interactable structures in *Coda*. It is a parent class of the Drum and is composed of the *Part* class. It contains the mutual definitions of these interactable structures in a general manner and invoke the functions according to the shared behaviour of the child classes like being played and so forth. For the envisaged version of *Coda*, the instrument to be implemented is determined as *Drums* but the implementation of the *Instrument* class will be made in an extendable way to allow easy addition of different types of instruments. Furthermore, possible user interactions will also be instrument specific.

#### 4.1.2.2 Drum Class

This class is a specialised version of *Instrument*. As explained above, *Drums* are the first specialized type of instrument to be implemented in the current version of *Coda*. This class will have an implementation specific to the real instrument itself. For the implementation of this class, domain experts will be consulted.

#### 4.1.2.3 Part Class

This class is a component class of *Instrument*. As explained above, instruments will be implemented in a way such that they will be composed of parts, some of which are interactable and some are stable. *Part*s will also be instrument specific and an instrument may be composed of a number of different or identical parts depending on the nature of the instrument. Encore, for the implementation of this class, domain experts will be consulted.

### 4.1.3 Play Layer

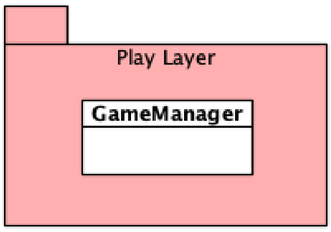

**Figure 6:** Subsystem Decomposition for the Play Layer of *Coda*

This layer is for handling the user interactions in *Coda*. Since this is a class within the *GamePlay Layer,* this class layer will be responsible from getting the recognized user gesture from the *HandGesture Layer* and applying the gesture accordingly. This layer will handle the interactions of the user with menus, with the instruments and the calibration screens. This layer will utilise data gathered from the other layers to determine the user action within *Coda*.

#### 4.1.3.1 GameManager Class

This class is responsible from managing the input and output data in this layer that will be utilised to provide the dynamics of the game. As explained above, user interactions will be handled to allow the user to interact with menus, calibration screens and instruments. In other words, user actions will be determined and applied by the help of this class.

### 4.1.4 View Layer

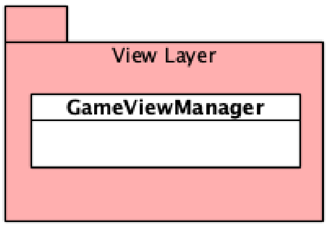

**Figure 7:** Subsystem Decomposition for the View Layer of *Coda*

This layer is for displaying the game view to the user as constructed by the *AR Layer* by communicating with it. Since this is a class within the *GamePlay Layer,* this class layer will be responsible from getting the constructed AR environment from the AR layer and displaying it on the screen of the smartphone. Mapping of the user gestures and the display coordinated will be made according to the representation in this layer as this layer represents what the user sees and interacts with.

#### 4.1.4.1 View Class

This class will represent what is displayed in the AR environment to the user. Coordinates and representation of the interactable structures are displayed as defined in this class. Therefore, mapping of the user interactions according to the gestures on the AR environment will be made according to what is stored in this class.

### 4.2 AR Layer

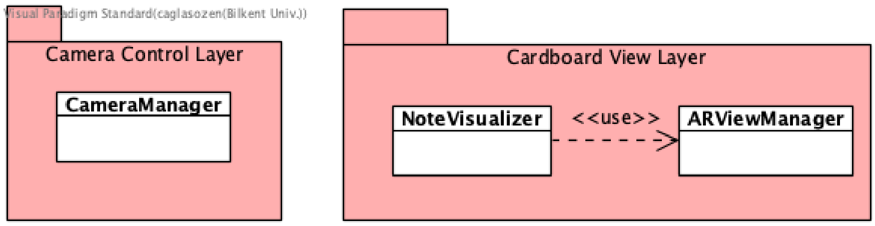

**Figure 8:** Subsystem Decomposition for the AR Layer of *Coda*

This layer is treated like the hardware layer of *Coda* in terms of display and video . Sub-layers of this layer handle camera input and the displaying of the AR environment as determined by the camera input, calibration and instrument structures. Furthermore, this layer handles the user visualization of notes that are instrument specific. Also, the AR view will be created by the input received from the *CameraControl* Layer and will be outputted by the *Cardboard View Layer.*

Functionalities and structures of these sublayers will be further discussed in the proceeding sub-sections.

#### 4.2.1 Camera Control Layer

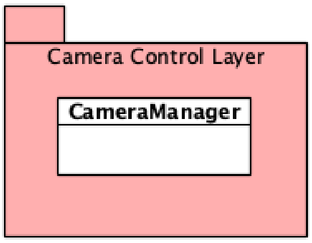

**Figure 8:** Subsystem Decomposition for the Camera Control Layer of *Coda*

This layer is for handling the camera input in *Coda*. Since this is a class within the *AR Layer,* this layer will be responsible from recording the real environment and passing it to the other layers for constructing the AR Layer as well as recording the hand gestures to be passed onto other classes for gesture recognition. This layer will handle capturing of hand gestures and environment. This layer will be an input only layer, meaning that it will only be utilised to take the input from the hardware device.

#### 4.2.1.1 CameraManager Class

This class is responsible from managing the camera as a hardware device. As explained above, camera input will be used in different layers. For appropriately capturing these inputs, *CameraManager Class* will be used.

### 4.2.2 Cardboard View Layer

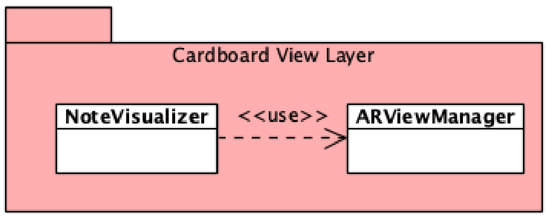

**Figure 9:** Subsystem Decomposition for the Cardboard View Layer of *Coda*

This layer is for displaying the game view to the user appropriately for the CardBoard constructed by the *AR Layer* by communicating with it. Since this is a class within the *AR Layer,* this layer will be responsible from constructing the AR View appropriately according to the state of the instruments, notes etc. Also, visualization of notes will be handles in this layer to construct a proper visualization of the notes correctly in the AR environment.

#### 4.2.2.1 NoteVisualizer Class

This class will be responsible from the visualization of different notes in an instrument specific manner. This class will be used by the *ARViewManager* class to determine what to construct in the AR environment for each note.

#### 4.2.2.2 ARViewManager Class

This class will construct what will be displayed in the AR environment to the user. It will receive the structures to be constructed in AR from other layers and construct the proper AR representations as an output to be displayed to the user.

## 4.3 Sound Layer

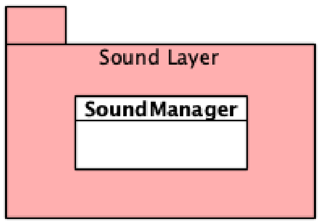

**Figure 9:** Subsystem Decomposition for the Sound Layer of *Coda*

This layer is for providing the audio experience to the user as fetched from memory in the case of songs and recordings and as defined in the instrument behaviours in the case of interactions with the instruments. This layer will handle interactions with the audio devices.This layer will be an output only layer, meaning that it will only be utilised to give auditory responses to the user.

### 4.3.1 SoundManager Class

This class is responsible from managing the speakers/ earphones as a hardware device. As explained above, auditory feedback will be given in several cases in Coda and is crucial for replicating the real-life experience of playing the instrument. For appropriately replicating the sounds of instruments and playing pieces, *SoundManager Class* will be used.

## 4.4 Storage Layer

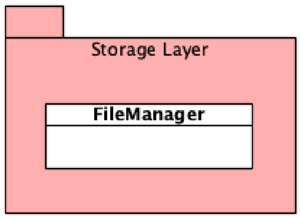

**Figure 10:** Subsystem Decomposition for the Storage Layer of *Coda*

This layer is for handling the I/O operations regarding writing and fetching from memory. This layer will handle interactions with the storage components for persistent data management. This layer will both receive inputs and give outputs, meaning that it will both write to the memory and read from the memory. Writing to the memory will be performed in cases of recordings and reading to the memory will be performed to play songs from the library or to play recorded pieces of the user.

### 4.4.1 FileManager Class

This class is responsible from managing the file operations on the local memory of the device as explained in *Section 3.4*. As explained above, recordings of the user will be written to the memory and they will be available for being played once written to the memory. Also, pre-loaded songs will be available to be played by default. In such cases, local memory and file systems of the device will be accessed by the *FileManager Class* and operations will be managed appropriately.

## 4.5 Hand Gesture Layer

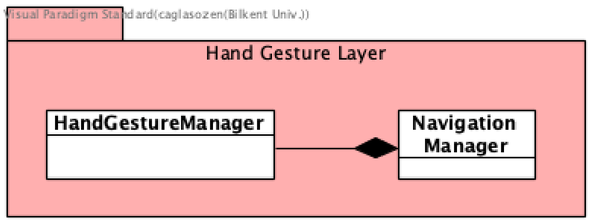

**Figure 11:** Subsystem Decomposition for the Hand Gesture Layer of *Coda*

This layer is for handling the hand gestures of the user and recognizing them to navigate in the app, receiving user input and interacting with the instruments. This layer will be responsible from recognizing the hand gestures of the user and classifying them as one of the defined gestures in the app according to the instrument being played or the part of the application being interacted with. Then the recognized and classified gestures, will be passed to other layers for applying the user interaction to the structures and providing the correct responses.

### 4.5.1 HandGestureManager Class

This class is responsible from recognizing and classifying the user gestures once received from the camera. As explained above, gestures will be instrument specific and possible gestures will be pre-determined within the app for each instrument individually and for all screens in the app.

### 4.5.2 NavigationManager Class

This class is used to use hand gestures to navigate in the application. For each screen, possible navigation gestures will be defined and the users will be able to navigate in the app according to their gestures once the gestures are recognized and classified by the *HandGestureManager* class.

## 5 New Knowledge Acquired and Learning Strategies Used

First and foremost, as mentioned in the previous documents we plan to learn about Image Processing in order to understand hand and finger gestures for enabling the user to interact with the instruments in the augment reality we render without the need for any other external tools like controllers. Since Image Processing might come short for detecting gestures with a non-constant background, and we will need to employ the help of deep learning. Hence we will learn about Computer Vision implemented by advanced Deep Learning and Image Analysis techniques. We plan to go over different implementations of such applications and writing our own simple code snippets which will give us the chance to gain hands-on knowledge.

As a novel approach, we’ve decided to learn Google’s MediaPipe for implementing the Computer Vision requirements of Coda for Hand Gesture Tracking. But since MediaPipe is not yet integrated with Unity directly, it is likely that we are going to have to reduce the use of Unity to the Graphics components of *Coda* only. For the rest, we are going to use Android SDK which we are already familiar with.

Another challenge for us will be programming an AR app that overlays 3D objects onto real world. Most of the current systems do not employ AR with VR Glasses/Cardboards, hence we will need to understand and learn how to reconstruct an environment to be able to put AR objects to a real environment.  We plan to follow along with online video tutorials and read from other online materials in order to learn about this topic. At this point, we might also perform interviews with experts from SimSoft which is a company experienced with AR and VR systems, by the help of our Innovation Expert.

If time permits, we would like to add certain cloud computing functionalities to our project for improving performance. In that case online learning will most probably be our primary source for learning the basics whereas doing exercises will be necessary for grasping the implementation details.

## 6 References

[1] Smith. B, “*New study demonstrates link between music and statistical learning*,” The Sydney Morning Herald, 2019. [Online]. Available: https://www.smh.com.au/technology/new-study-demonstrates-link-between-music-and-statistical-learning-20170514-gw4eec.html. [Accessed: Oct. 10, 2019].

[2] ABRSM, *ABRSM:*. [Online]. Available: https://es.abrsm.org/en/making-music/4-the-statistics/. [Accessed: Oct. 10, 2019].

[3] “Size of average song in MB,” *Spotify.Community*, 05-Sep-2015. [Online]. Available: https://community.spotify.com/t5/Android/Size-of-average-song-in-MB/td-p/1200704. [Accessed: Dec. 27, 2019].

[4] “How long can a voice recorder on an Android record?,” *Quora*, 03-Jun-2018. [Online]. Available: https://www.quora.com/How-long-can-a-voice-recorder-on-an-Android-record. [Accessed: Dec. 27, 2019].