Software Design Document
for

watch & touch

by DialecTech
Giray Havur (1630870)
Melike Ercan (1560135)
Utku Şirin (1560838)
Yaman Umuroğlu (1560614)
Table of Contents

1. Introduction......................................................................................................................... 3
   1.1. Problem Definition ........................................................................................................ 3
   1.2. Purpose ......................................................................................................................... 3
   1.3. Scope ............................................................................................................................ 3
   1.4. Overview .................................................................................................................... 3
   1.5. Definitions, Acronyms and Abbreviations.................................................................. 4
   1.6. References................................................................................................................... 5

2. System Overview .................................................................................................................. 6
   2.1. The Interactive Whiteboard Client ............................................................................... 7
   2.2. The Collaboration Client ............................................................................................. 7

3. Design Considerations ......................................................................................................... 7
   3.1. Design Assumptions, Dependencies and Constraints.................................................... 7
       3.1.1. Interactive Whiteboard Client ............................................................ 7
       3.1.2. Collaborativity Client: ............................................................... 8
   3.2. Design Goals and Guidelines ..................................................................................... 8

4. Data Design ........................................................................................................................ 9
   4.1. Data Description .......................................................................................................... 9
       4.1.1. Description of data entities ........................................................................... 9
       4.1.2. Annotation data storage ............................................................................. 18
       4.1.3. Supported content file formats ................................................................. 19
   4.2. Data Dictionary ......................................................................................................... 20

5. System Architecture .......................................................................................................... 23
   5.1. Architectural Design .................................................................................................... 23
       5.1.1. Architecture of Interactive Whiteboard Client ....................................... 23
       5.1.2. Architecture of Collaboration Client ...................................................... 25
   5.2. Description of Modules ............................................................................................... 25
       5.2.1. TaskManagement module ....................................................................... 26
       5.2.2. ContentDisplay module ........................................................................... 27
       5.2.3. Drawing module ......................................................................................... 28
       5.2.4. Sessions module ....................................................................................... 30
       5.2.5. WiimoteInput module ............................................................................. 31
       5.2.6. Configuration module .............................................................................. 33

6. User Interface Design ......................................................................................................... 34
   6.1. Overview of User Interface ........................................................................................ 34
       6.1.1. User Interfaces for the IWBC .................................................................. 34
       6.1.2 User Interfaces for the Collaborativity Client: ............................................ 46
   6.2. Screen Images ............................................................................................................. 50
   6.3. Screen Objects and Actions ...................................................................................... 51

7. Detailed Design .................................................................................................................. 52
   7.1. Detailed Module Description ...................................................................................... 52
       7.1.1. TaskManagement module ........................................................................ 53
       7.1.2. ContentDisplay module ............................................................................ 55
       7.1.3. Drawing module ........................................................................................... 57
       7.1.4. Sessions module ............................................................................................. 59
       7.1.5. WiimoteInput and Configuration modules .................................................. 61
   7.2. Detailed design for the Collaboration Client .............................................................. 64
8. Libraries and Tools................................................................................................................................................. 65
  8.1 Qt........................................................................................................................................................................ 65
    8.1.1. Description................................................................................................................................................. 65
    8.1.2. Usage in Watch & Touch......................................................................................................................... 66
  8.2 Wiiuse ................................................................................................................................................................. 66
    8.2.1. Description................................................................................................................................................. 66
    8.2.2. Usage in Watch & Touch......................................................................................................................... 66
  8.3 Google Documents API....................................................................................................................................... 66
    8.3.1. Description................................................................................................................................................. 66
    8.3.2. Usage in Watch & Touch......................................................................................................................... 67
  8.4 Screencasting tools............................................................................................................................................ 67
    8.4.1. Description................................................................................................................................................. 67
    8.4.2. Usage in Watch & Touch......................................................................................................................... 67
  8.5. Poppler ............................................................................................................................................................. 67
    8.5.1. Description................................................................................................................................................. 67
    8.5.2. Usage in Watch & Touch......................................................................................................................... 68
9. Time Planning (Gantt Chart) ............................................................................................................................... 69
  9.1. Term 1 Gantt Chart .......................................................................................................................................... 69
  9.2. Term 2 Gantt Chart .......................................................................................................................................... 70
10. Conclusion ......................................................................................................................................................... 71
11. Appendix .......................................................................................................................................................... 71
    11.1. Sequence Diagrams ..................................................................................................................................... 71
      11.1.1. Wiimote Initialization sequence diagram............................................................................................ 71
      11.1.2. Wiimote Calibration sequence diagram ............................................................................................ 72
      11.1.3. Content Display and Annotation sequence diagram ......................................................................... 72
      11.1.4. Sketching sequence diagram ............................................................................................................. 73
      11.1.5. Collaborative Drawing overview sequence diagram .......................................................................... 74
      11.1.6. Classroom Session sequence diagram .............................................................................................. 75
    11.2. Detailed Use-Case Diagrams ........................................................................................................................ 76
      11.2.1. Calibration use-case diagram ............................................................................................................. 76
      11.2.2. Log in and Log out use-case diagrams .............................................................................................. 76
      11.2.3. Create Student Account and Instructor Account use-case diagrams ............................................. 77
      11.2.4. Annotation use-case diagram ............................................................................................................ 78
      11.2.5. Collaboration use-case diagram ........................................................................................................ 79
      11.2.6. File Manipulation use-case diagram .................................................................................................. 89
      11.2.7. Presentation Manipulation use-case diagram ..................................................................................... 90
      11.2.8. Video Manipulation use-case diagram ............................................................................................. 91
      11.2.9. Webpage Manipulation .................................................................................................................... 92
    11.3. Activity diagrams ....................................................................................................................................... 84
      11.3.1. IWBC activity diagrams.................................................................................................................... 84
      11.3.2. CBC activity diagrams..................................................................................................................... 95
1. Introduction

Initial Software Design Document of Watch & Touch provides the necessary definitions to conceptualize and further formalize the design of the software, of which its requirements and functionalities were summarized in the previous requirements analysis report. The report was written according to the guidelines provided in the IEEE Recommended Practice for Software Design Descriptions[1]. The aim is to provide a guide to a design that could be easily implemented by any designer reading this report.

1.1. Problem Definition

The aim of Watch & Touch is to create a complete, free & open source, multi-platform, interactive whiteboard system which supports multi-touch gestures, complemented by an in-classroom collaborative working client to increase student participation in classroom activities. Hardware and software solutions targeting these have existed for many years, especially as an integrated part of “smart class” solutions, but deployment is not widespread (only 2 smart classes in METU) due to their high cost (around a thousand dollars[2] for interactive whiteboard units not including the projector, and much higher costs for complete smart class solutions). A review of existing solutions can be found in the 1.4. User and Literature Survey section of the Watch & Touch SRS[2]. Considering existing solutions or methods, it is desired to create a complete, free & open source, multi-platform, interactive blackboard system that can be utilized at the relatively low cost of its available off-the-shelf hardware components, as well as an in-classroom collaborative working environment to complement the classroom activity when the necessary hardware is available.

1.2. Purpose

In the Watch & Touch SRS[2] document, Watch & Touch’s desired features and requirements were stated. This SDD is intended to provide a software system design which will satisfy the given functional and non-functional requirements in line with the provided assumptions and constraints. As the design of the system is the core of its development, the document is intended to be viewed primarily by the DialecTech team which will be developing Watch & Touch, and will be presented to the Department of Computer Engineering faculty as part of the Senior Computer Engineering Design (CENG491) course.

1.3. Scope

Scope of ISDD is to provide information about design of the project. This document covers the architectural design, data design, procedural design, design constraints, development schedule. Also hardware and software requirements and working environment will be explained.
1.4. Overview

This document provides information about Watch & Touch software system. As it is explained in section 2, the system consists of two separate but interconnected software applications, named the Interactive Whiteboard Client (IWBC) and the Collaboration Client (CBC). This document explains these two components in detail. In section 3, design issues such as assumptions, dependencies and constraints, design goals and guidelines that are used is explained. In section 4, information about data design and data entities in the system is provided. Section 5 is about the system’s overall architecture; the architectural design of the software applications and detailed description of modules are explicated. Section 6 provides design details on the user interfaces. In section 8, time planning and scheduling issues will be demonstrated by a Gantt Chart. The document is ended with a conclusion that will be given in section 10. Additional UML diagrams are provided for further clarification in section 11.

1.5. Definitions, Acronyms and Abbreviations

Definitions, Acronyms and Abbreviations in Watch & Touch Software are explained in the following table.

<table>
<thead>
<tr>
<th>Wiimote</th>
<th>The hardware controller designed by Nintendo for use with the Nintendo Wii gaming console, that offers the tracking of movements of up to 4 infrared points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watch &amp; Touch</td>
<td>The software system hereby documented, including the IWBC and CBC clients</td>
</tr>
<tr>
<td>machine</td>
<td>A computing device, used either by the instructor or one of the students, that fulfills the Watch &amp; Touch requirements</td>
</tr>
<tr>
<td>IWB</td>
<td>Interactive Whiteboard, the interactive projected display created using the Watch &amp; Touch IWBC</td>
</tr>
<tr>
<td>IWBC</td>
<td>Interactive Whiteboard Client, the piece of software installed on the instructor’s machine which allows interaction on the projected display using the IR pen and the Wiimote, and provides annotation features.</td>
</tr>
<tr>
<td>CBC</td>
<td>The Watch &amp; Touch Collaboration Client, the piece of software installed on student machines that allows them to work collaboratively</td>
</tr>
<tr>
<td>collaborative working</td>
<td>A process in which several users are able to simultaneously make modifications to a common document and see the modifications done by others</td>
</tr>
<tr>
<td>collaboration hub</td>
<td>The machine which will serve as a hub for in-classroom collaboration, which is determined as the instructor’s machine</td>
</tr>
<tr>
<td>annotation</td>
<td>The ability to create user content such as hand-made drawings, shapes or</td>
</tr>
<tr>
<td>Gesture</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>multi-touch</td>
<td>A predefined movement involving multiple points of interaction</td>
</tr>
<tr>
<td>pinch gesture</td>
<td>A multi-touch gesture in which two points along a line move towards, or away</td>
</tr>
<tr>
<td>rotation gesture</td>
<td>A multi-touch gesture in which two points rotate clockwise or counterclockwise along a fixed axis</td>
</tr>
<tr>
<td>swipe gesture</td>
<td>A multi-touch gesture in which two points move towards the same horizontal direction</td>
</tr>
<tr>
<td>IR pen</td>
<td>Infrared pen, a pen-shaped object whose tip radiates Wiimote-detectable infrared light when desired</td>
</tr>
<tr>
<td>IR ring</td>
<td>An IR pen that can be worn on a finger, several can be used multi-touch gestures</td>
</tr>
<tr>
<td>content</td>
<td>Visual content in presentation, video or webpage form.</td>
</tr>
<tr>
<td>SDD</td>
<td>Software Design Document</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
</tbody>
</table>

1.6. References


A common software engineering standard to provide some guidance and recommended approaches for specifying software design descriptions

[2] Software Requirements Specification for Watch & Touch

The SRS document for Watch & Touch, prepared according to the IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications

[3] The KISS Principle in design process

[4]
2. System Overview

Watch & Touch consists mainly of two separate but interconnected software applications, named the Interactive Whiteboard Client (IWBC) and the Collaboration Client (CBC).
2.1. The Interactive Whiteboard Client

The IWBC is an application that is capable of turning any projected surface into a multi-touch-input whiteboard. In terms of providing interaction, it calculates the position of the infrared pen on the surface (by utilizing the input from a Wiimote) in terms of screen coordinates and generates mouse events accordingly to provide interaction on the selected point, as well as interpret the movements of multiple IR sources as multi-touch gestures to enhance the general user experience. An important consequence of this capability is the annotation features provided - the instructor is able to display presentations, web pages and videos from inside the IWBC and take handwritten notes on top of these visual content when desired. The IWBC also allows the instructor to actively partake in the classroom collaboration activities, in a way similar to the facilities provided to the students by the CBC.

2.2. The Collaboration Client

The CBC is a smaller application intended for use by students on their own machines, and allows them to engage in classroom collaboration by creating collaborative drawings. The CBC establishes collaborative links to other machines which also have . The collaborative link exists not only between students but also between students and the instructor; students can see the information on the instructor’s projected screen on their own machines, and (with the instructor’s permission) make drawings on the content seen on the instructor’s screen, to ask questions or express themselves by visually emphasizing or annotating the currently displayed content.

3. Design Considerations

Special design issues which need to be addressed or resolved before attempting to devise a complete design solution are noted here.

3.1. Design Assumptions, Dependencies and Constraints

Watch & Touch is assumed to operate under the presence of certain factors with regard to hardware, system software and general properties of the operating environment. These factors are stated, separately for the IWBC and the CBC, in the following two subsections 3.1.1. and 3.1.2.

3.1.1. Interactive Whiteboard Client

- An operational Wiimote is assumed to be present and connected via Bluetooth to the instructor’s machine
- The IWBC must be installed and successfully launching on the instructor’s machine
- At least one IR pen with the necessary characteristics for being detected by the Wiimote must be present and functional. For multi-touch gestures, at least two such IR sources are required.
- A flat projected surface is required to be turned into an interactive whiteboard.
• The Wiimote must be positioned in such a way that all four corners of the projected display are within the line of sight of the Wiimote infrared camera.

3.1.2. Collaborativity Client:
• At least several student machines should be present. The ideal would be one machine per student, however this is not a requirement.
• The Watch & Touch collaborativity client must be installed and successfully launching on all the machines intended for collaboration
• The machines intended for collaboration must be connected by a Local Area Network, including the instructor’s machine.
• The machines must have the necessary two-dimensional input hardware interface (such as a mouse or touch surface) that gives them the capability to provide the input for drawing

3.2. Design Goals and Guidelines
• Both software applications in the system (IWBC and CBC) must be capable of working with full functionality on multiple platforms and operating systems. Towards this end, the Qt framework will be used in the development of the project. Microsoft Windows XP® and Ubuntu 10.04 have been chosen as the testing platforms due to their widespread use[2].
• Since Watch & Touch is heavily centered on its user interfaces:
  ○ the user interfaces must be consistent, clean and easy to use in general
  ○ the Cornell University Ergonomic Guidelines for User-Interface Design[7] will serve as the guideline during the user interface design process
  ○ existing Qt widgets providing UI functionality should be preferred as much as possible instead of re-creating user interface components, as this will consume more time and be less stable
  ○ where possible, a smaller degree of abstraction should be preferred between the underlying business logic and the user interface, to keep a simpler software structure
• During the design process, the KISS principle will serve as the primary guideline for Watch & Touch. The KISS principle[8] states that simplicity should be a key goal in design, and that unnecessary complexity should be avoided.
• During design and development, the principles mentioned in the Designing Qt Style C++ APIs[4] document will be followed to obtain source code which is easier to understand for developers, and more suitable for Qt’s style.
• To keep Watch & Touch as free and open[6] as possible, use of external libraries and tools which do not offer LGPL or GPL should be avoided.
• The Scalable Vector Graphics (SVG) format[8] should be used to store annotation and drawing data, for the following reasons:
  ○ Qt has good built-in support for SVG
  ○ vector graphics allows for easier processing of individual annotation elements
○ scaling the annotations does not cause any loss of quality
○ format is suitable for extensions and metadata storage since it is XML-based

4. Data Design

4.1. Data Description

This section contains a description of the information domain for Watch & Touch. First, a description of the structure of defined data entities is provided, followed by two sections providing an overview of how the annotation storage and content display work.

Watch & Touch does not make extensive use of relational data, thus no databases are utilized. In fact the only persistent data is for the annotations (whose storage is described in section 4.1.2) and the configuration data (4.1.8).

4.1.1. Description of data entities

4.1.1.1. AnnotationBase data entity

The AnnotationBase data entity describes the basic content of an annotation and forms the base of the other *Annotation data entities.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParentID</td>
<td>String</td>
<td>The MD5 hash of the parent content</td>
</tr>
<tr>
<td>Geometry</td>
<td>Rectangle</td>
<td>Position and size of annotation</td>
</tr>
<tr>
<td>LayerCount</td>
<td>Integer</td>
<td>Number of items in the Layers field</td>
</tr>
<tr>
<td>Layers</td>
<td>Array of SVGData</td>
<td>Annotation SVG data is contained in these items</td>
</tr>
</tbody>
</table>

4.1.1.2. PresentationAnnotation data entity

The PresentationAnnotation data entity describes the annotation on a single presentation slide.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlideNumber</td>
<td>Integer</td>
<td>Which slide the annotation belongs to</td>
</tr>
</tbody>
</table>
Data | AnnotationBase | Base annotation data entity

### 4.1.1.3. WebpageAnnotation data entity
The WebpageAnnotation data entity describes the annotation on a single webpage anchor.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnchorName</td>
<td>String</td>
<td>A string identifying the webpage anchor on which the annotation was made</td>
</tr>
<tr>
<td>Data</td>
<td>AnnotationBase</td>
<td>Base annotation data entity</td>
</tr>
</tbody>
</table>

### 4.1.1.4. VideoAnnotation data entity
The VideoAnnotation data entity describes the annotation on a single video frame.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimelinePosition</td>
<td>String</td>
<td>The position in the video timeline to which the annotation belongs (hh:mm:ss)</td>
</tr>
<tr>
<td>Data</td>
<td>AnnotationBase</td>
<td>Base annotation data entity</td>
</tr>
</tbody>
</table>

### 4.1.1.5. CollaborativeSession data entity
The CollaborativeSession data entity describes a group of participants working on a collaborative drawing.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParticipantCount</td>
<td>Integer</td>
<td>Number of participants in the session</td>
</tr>
<tr>
<td>Participants</td>
<td>Array of CollaborativeParticipant</td>
<td>Information on participants in the session</td>
</tr>
<tr>
<td>Field Name</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SessionName</td>
<td>String</td>
<td>The name of the session, visible to all</td>
</tr>
<tr>
<td>SessionPassword</td>
<td>String</td>
<td>MD5 hash of session password</td>
</tr>
<tr>
<td>TheDrawing</td>
<td>CollaborativeDrawing</td>
<td>The collaborative drawing being created by this session</td>
</tr>
<tr>
<td>CreationDate</td>
<td>DateTime</td>
<td>When the session was created</td>
</tr>
</tbody>
</table>

### 4.1.1.6. CollaborativeParticipant data entity

The CollaborativeParticipant data entity describes a participant in the collaborative drawing session.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NameSurname</td>
<td>String</td>
<td>Publically visible name and surname of participant</td>
</tr>
<tr>
<td>EMail</td>
<td>String</td>
<td>E-mail address of the participant</td>
</tr>
<tr>
<td>IsInstructor</td>
<td>Boolean</td>
<td>Whether the participant has instructor privileges</td>
</tr>
</tbody>
</table>

### 4.1.1.7. CollaborativeDrawing data entity

The CollaborativeDrawing data entity describes a single collaborative drawing.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>CollaborativeSession</td>
<td>The collaborative drawing session which is creating this drawing.</td>
</tr>
<tr>
<td>LastModified</td>
<td>DateTime</td>
<td>When the drawing was last modified</td>
</tr>
<tr>
<td>LayerCount</td>
<td>Integer</td>
<td>Number of layers in the drawing</td>
</tr>
<tr>
<td>Layers</td>
<td>Array of SVGData</td>
<td>Actual drawing SVG data in layers</td>
</tr>
</tbody>
</table>
4.1.1.8. Configuration data entity
The Configuration data entity describes the current configuration of the IWBC. It is made persistent in XML form, where the field names serve as tag names.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiimoteData</td>
<td>WiimoteConfig</td>
<td>Information pertaining to the currently connected Wiimote.</td>
</tr>
<tr>
<td>ScreencastingBackend</td>
<td>String</td>
<td>Which screencasting backend is to be used (&quot;ffmpeg&quot; or &quot;vlc&quot;)</td>
</tr>
<tr>
<td>GestureData</td>
<td>Array of GestureMap</td>
<td>The matching of gestures to actions</td>
</tr>
</tbody>
</table>

4.1.1.9. WiimoteConfig data entity
The Configuration data entity describes the current configuration of the Wiimote controller. It is made persistent in XML form, where the field names serve as tag names.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UniqueID</td>
<td>Integer</td>
<td>The unique identifier assigned to the Wiimote by the wiiuse library</td>
</tr>
<tr>
<td>CalibrationData</td>
<td>Array of (Float, Float)</td>
<td>The raw IR coordinates of the four IR calibration points received</td>
</tr>
</tbody>
</table>

4.1.1.10. GestureMap data entity
The GestureMap data entity describes a single gesture-action mapping. It is made persistent in XML form, where the field names serve as tag names.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GestureID</td>
<td>GestureEnum</td>
<td>The identifier of the predefined gesture</td>
</tr>
</tbody>
</table>
### 4.1.11. GestureEnum enumeration

This data type describes and enumerates the possible multi-touch gestures recognized by Watch & Touch.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TapAndHoldGesture</td>
<td>User holds keeps pointer in the same location for &gt;1 sec</td>
</tr>
<tr>
<td>1</td>
<td>SwipeLeft</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SwipeUp</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SwipeRight</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SwipeDown</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PinchIn</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PinchOut</td>
<td></td>
</tr>
</tbody>
</table>

4.1.11.0: TapAndHoldGesture
4.1.1.1.1: SwipeLeft

4.1.1.1.2: SwipeUp

4.1.1.1.3: SwipeRight
4.1.1.11.4: SwipeDown

4.1.1.11.5: PinchIn

4.1.1.11.6: PinchOut
4.1.1.12. ActionEnum enumeration
This data type describes and enumerates the possible actions which can be associated with gestures in Watch & Touch.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OpenMenu</td>
<td>Open the context menu for the current task</td>
</tr>
<tr>
<td>1</td>
<td>MenuShow</td>
<td>Show the global menu bar</td>
</tr>
<tr>
<td>2</td>
<td>MenuHide</td>
<td>Hide the global menu bar</td>
</tr>
<tr>
<td>3</td>
<td>Undo</td>
<td>Perform undo on the current task</td>
</tr>
<tr>
<td>4</td>
<td>Redo</td>
<td>Perform redo on the current task</td>
</tr>
<tr>
<td>5</td>
<td>CloseTask</td>
<td>Close the current task</td>
</tr>
<tr>
<td>6</td>
<td>ZoomIn</td>
<td>Zoom in to the current task</td>
</tr>
<tr>
<td>7</td>
<td>ZoomOut</td>
<td>Zoom out of the current task</td>
</tr>
</tbody>
</table>

4.1.1.13. TaskType enumeration
This data type describes and enumerates the possible types of Watch & Touch IWBC tasks.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Configuration</td>
<td>Task for configuring Watch &amp; Touch</td>
</tr>
<tr>
<td>1</td>
<td>Sketching</td>
<td>Task for doing freehand sketches</td>
</tr>
<tr>
<td>2</td>
<td>PresentationDisplay</td>
<td>Task for displaying and annotating presentations</td>
</tr>
<tr>
<td>3</td>
<td>WebpageDisplay</td>
<td>Task for displaying and annotating webpages</td>
</tr>
<tr>
<td>4</td>
<td>VideoDisplay</td>
<td>Task for displaying and annotating videos</td>
</tr>
<tr>
<td>5</td>
<td>SessionManagement</td>
<td>Task for creating and joining collaborative drawing sessions</td>
</tr>
<tr>
<td>6</td>
<td>CollaborativeDrawing</td>
<td>Task for working collaboratively on a drawing</td>
</tr>
</tbody>
</table>

4.1.1.14. TaskStatus enumeration
This data type describes and enumerates the possible statuses of Watch & Touch IWBC tasks.
<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NotVisible</td>
<td>Task window is not visible / minimized</td>
</tr>
<tr>
<td>1</td>
<td>VisibleNormal</td>
<td>Task window is visible but not fullscreen</td>
</tr>
<tr>
<td>2</td>
<td>VisibleFullscreen</td>
<td>Task window is visible fullscreen</td>
</tr>
<tr>
<td>3</td>
<td>CloseRequest</td>
<td>Task window has asked to be closed</td>
</tr>
</tbody>
</table>

### 4.1.15. ContentType enumeration
This data type describes and enumerates the possible types of Watch & Touch visual content.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Presentation</td>
<td>Content type is PDF, PPT or ODP</td>
</tr>
<tr>
<td>1</td>
<td>Webpage</td>
<td>Content type is HTML webpage</td>
</tr>
<tr>
<td>2</td>
<td>Video</td>
<td>Content type is Phonon-supported video</td>
</tr>
</tbody>
</table>

### 4.1.16. GoogleDocsEntry data entity
The GoogleDocsEntry data entity describes a single Google Document

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentTitle</td>
<td>String</td>
<td>Title of the Google document</td>
</tr>
<tr>
<td>ResourceID</td>
<td>String</td>
<td>Unique identifier for this Google document</td>
</tr>
</tbody>
</table>

### 4.1.17. DrawingMode enumeration
This data type describes and enumerates the possible types of Watch & Touch drawing modes.

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Freehand</td>
<td>Freehand drawing without restrictions</td>
</tr>
</tbody>
</table>
### 4.1.2. Annotation data storage

This section gives a general description of how the annotations made on visual content are stored and retrieved.

#### 4.1.2.1. Storage logic

The matching of content and annotations is made through a unique identifier. Each visual content is has a unique identifier, which represents the current state of the content file. This identifier is generated by running the MD5 algorithm[^9] on the content file. This process ensures that the identifier is not modified by trivial changes such as changing the content filename, and prevents mismatches between the annotation and the content file in case the content changes.

All annotations created by the IWBC are stored under the `annotations/` directory, located under the IWBC application directory. For each content which is annotated by the user, a sub-directory under `annotations/` is created. This sub-directory is named as the unique identifier of the annotated content, and inside it are all the annotation data files which belong to this content.

Each annotation data file contains the annotation made in a different part of the content, and is named according to the following convention:

\[
\text{annotation}_{\%\text{subcontextid}}_{\%\text{layerno}}.\text{wta}
\]

The `%subcontextid` identifies which part of the content the annotation is related to. It is generated differently for different content types:

- **Presentations**: slide number, starting from 0
- **Webpages**: anchor name (such as `#section1`) if exists, nothing otherwise
- **Videos**: timeline position (hh:mm:ss) in the video

[^9]: MD5 algorithm
The \%layerno\ identifies the layer number of the annotation. Currently, all annotation data is stored in layer number 0; if future expansion layers are needed (which will be numbered 1, 2, 3...) they will be stacked in front of layer 0.

### 4.1.2.2. File format

All annotations will be stored using the Scalable Vector Graphics (SVG) format, an XML-based file format for describing two-dimensional vector graphics\(^8\). See section 3.2 for the reasons why SVG was chosen as the annotation storage format.

Since the actual vector data will be generated using the Qt’s QSvgGenerator and rendered using QSvgRenderer, no further details will be provided here as to how the annotation vector data is generated.

### 4.1.3. Supported content file formats

This section provides a list of the supported content file formats for presentations, videos and webpages. As long as the general content format is preserved (numbered slides for presentations, vertical scrolling for webpages and a hour/minute/second partitioned timeline for videos), support for other file formats can be added simply by adding the corresponding viewer code.

#### 4.1.3.1. Presentations

Portable Document Format (PDF) will be the primary supported format for presentations. These files will be displayed using the Poppler Qt4 interface library\(^{10}\). Since the viewing is done by this external library, no further details on the PDF file format structure are provided here.

Additionally, support for Microsoft PowerPoint and OpenDocument Presentation formats should also be provided if time allows. This support can be provided by converting these formats to PDF via some command line tool.

#### 4.1.3.2. Videos

The video playback functionality will be provided by the Phonon multimedia framework\(^{11}\) which is a part of Qt. All seek-enabled video formats which is supported by the current Phonon backend are supported. Since the video display is handled by Phonon, no further details on the video file format structures are provided here.

#### 4.1.3.3. Webpages

The displaying of webpages will be done using the integrated WebKit support\(^{12}\) in Qt; thus, all HTML subformats supported by WebKit are also supported. The webpages are assumed to be only vertical-scrolling (horizontal scrolling is not supported) and static in terms of geometry (interacting with links etc.
does not change the width/height of the webpage). Since the webpage display is handled by WebKit, no further details on the HTML file format structure are provided here.

### 4.2. Data Dictionary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Refer to Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionType</td>
<td>data entity</td>
<td>4.1.1.12</td>
</tr>
<tr>
<td>AnnotationBase</td>
<td>data entity</td>
<td>4.1.1.1</td>
</tr>
<tr>
<td>AnnotationWidget</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>BaseDrawingWidget</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>ClassroomSession</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>CollaborativeDrawing</td>
<td>data entity</td>
<td>4.1.1.7</td>
</tr>
<tr>
<td>CollaborativeDrawingTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>CollaborativeDrawingWidget</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>CollaborativeParticipant</td>
<td>data entity</td>
<td>4.1.1.6</td>
</tr>
<tr>
<td>CollaborativeSession</td>
<td>data entity</td>
<td>4.1.1.5</td>
</tr>
<tr>
<td>Configuration</td>
<td>data entity</td>
<td>4.1.1.8</td>
</tr>
<tr>
<td>Configuration</td>
<td>module</td>
<td>7.1.5</td>
</tr>
<tr>
<td>ConfigurationData</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>ConfigurationManager</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>ConfigurationTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>ContentDisplay</td>
<td>module</td>
<td>5.2.2</td>
</tr>
<tr>
<td>ContentDisplay</td>
<td>module</td>
<td>7.1.2</td>
</tr>
<tr>
<td>ContentDisplayTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>ContentMatcher</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>ContentSelector</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>ContentType</td>
<td>data entity</td>
<td>4.1.1.15</td>
</tr>
<tr>
<td>Class/Module</td>
<td>Type</td>
<td>Module/Version</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Drawing</td>
<td>module</td>
<td>7.1.3</td>
</tr>
<tr>
<td>DrawingMode</td>
<td>data entity</td>
<td>4.1.1.17</td>
</tr>
<tr>
<td>DrawingData</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>EmailSender</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>EventGenerator</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>FileSystemBrowser</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>GestureEnum</td>
<td>data entity</td>
<td>4.1.1.11</td>
</tr>
<tr>
<td>GestureMap</td>
<td>data entity</td>
<td>4.1.1.10</td>
</tr>
<tr>
<td>GestureMapper</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>GoogleDocsAccess</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>GoogleDocsEntry</td>
<td>data entity</td>
<td>4.1.1.16</td>
</tr>
<tr>
<td>IContentSelection</td>
<td>interface</td>
<td>7.1.2</td>
</tr>
<tr>
<td>IDisplayedContent</td>
<td>interface</td>
<td>7.1.2</td>
</tr>
<tr>
<td>InputCalibration</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>InputReceiver</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>MessageTransciever</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>PresentationAnnotation</td>
<td>data entity</td>
<td>4.1.1.2</td>
</tr>
<tr>
<td>PresentationDisplayWidget</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>PresentationDisplayTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>RecentlyUsed</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>ScreenCapture</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>Screencasting</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>Session</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>SessionManagementTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>SessionManager</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>Sessions</td>
<td>module</td>
<td>7.1.4</td>
</tr>
<tr>
<td>Class/Module</td>
<td>Type</td>
<td>Section</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>SessionUser</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>SketchingTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>SketchingWidget</td>
<td>class</td>
<td>7.1.3</td>
</tr>
<tr>
<td>SnapshotRetriever</td>
<td>class</td>
<td>7.1.4</td>
</tr>
<tr>
<td>Task</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>TaskManagement</td>
<td>module</td>
<td>7.1.1</td>
</tr>
<tr>
<td>TaskManager</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>TaskStatus</td>
<td>data entity</td>
<td>4.1.1.14</td>
</tr>
<tr>
<td>TaskType</td>
<td>data entity</td>
<td>4.1.1.13</td>
</tr>
<tr>
<td>VideoAnnotation</td>
<td>data entity</td>
<td>4.1.1.4</td>
</tr>
<tr>
<td>VideoDisplayWidget</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>VideoDisplayTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>WebpageAnnotation</td>
<td>data entity</td>
<td>4.1.1.3</td>
</tr>
<tr>
<td>WebpageDisplayWidget</td>
<td>class</td>
<td>7.1.2</td>
</tr>
<tr>
<td>WebpageDisplayTask</td>
<td>class</td>
<td>7.1.1</td>
</tr>
<tr>
<td>WiimoteConfig</td>
<td>data entity</td>
<td>4.1.1.9</td>
</tr>
<tr>
<td>WiimoteInput</td>
<td>module</td>
<td>7.1.5</td>
</tr>
<tr>
<td>WiimoteManager</td>
<td>class</td>
<td>7.1.5</td>
</tr>
<tr>
<td>wiiuse</td>
<td>library</td>
<td>7.1.5</td>
</tr>
</tbody>
</table>
5. System Architecture

A general description of the Watch & Touch software system architecture is presented in the following sections, in a top-down manner.

5.1. Architectural Design

Watch & Touch will include two separate but related software applications, the Interactive Whiteboard Client and the Collaboration Client whose functional descriptions and requirements are provided in the sections 2.2 and 3.2.1 of Watch & Touch SRS\(^2\). The deployment diagram below illustrates the topmost level view of Watch & Touch architecture by presenting a view of how the applications will be deployed in the usage environment.

![Deployment Diagram](image)

*Figure 5.1.1: Deployment diagram for Watch & Touch, showing the physical positioning of the IWB and CBC applications on instructor and student machines*

5.1.1. Architecture of Interactive Whiteboard Client

The IWBC, which provides the primary interactive whiteboard-related functionality such as content display and annotation, as well as collaboration-related functionality, will be composed of six top-level modules, each formed of components grouped according to their functional similarity. In the following subsections of this section, a brief overview of the responsibilities of each of these modules is provided. A simplified overview of the IWBC architecture (simplified: not all components and associations inside the modules are shown - consult the following subsections for more detailed views of the internal module structures) intending to show how these modules are connected together is presented in the following component diagram.
5.1.2. Architecture of Collaboration Client

The CBC, which basically provides collaboration-related functionality, will be composed of three top-level modules, each formed of components grouped according to their functional similarity. In the following subsections of this section, a brief overview of the responsibilities of each of these modules is provided. A simplified overview of the CBC architecture (simplified: not all components and associations inside the modules are shown - consult the following subsections for more detailed views of the internal module structures) intending to show how these modules are connected together is presented in the following component diagram.

![Component Diagram](image)

Figure 5.2.1.1: Simplified component diagram for the Watch and Touch Collaboration Client

As can be noted from the above diagram, the CBC architecture is essentially a minimized subset of the IWBC architecture - the same components are reused with a lesser degree of functionality, thus a separate module description for the CBC is not provided.

5.2. Description of Modules

Each component described in these modules may correspond to one or more classes - see chapter 7 for class descriptions.
5.2.1. TaskManagement module

![Component diagram for the TaskManagement module]

5.2.1.1. Processing narrative
The TaskManagement module is responsible for the logical and visual management of the instances of multiple tasks which the IWBC is capable of - creating new task instances, switching between and terminating existing instances, and forming a basis for the user interface elements related with these tasks - the TaskManager is responsible for the IWBC Main Menu and each QWidget-derived *Task component is responsible for its own user interface elements.

5.2.1.2. Interface description
All tasks allow performing the user interface related tasks (set window geometry, minimize, maximize, show/hide...) and is realized by the QWidget set of member functions with which the component is associated.

5.2.1.3. Processing detail
The TaskManager component creates new *Task instances, stores the existing instances in an internal structure, and performs the necessary operations on them when requested through the user interface.
5.2.1.4. Dynamic behavior

Each *Task component is associated with a QWidget-derived component, which realizes the user interface and specific functionality for that particular task. The mapping is as follows:

- PresentationDisplayTask to ContentDisplay::PresentationDisplay
- VideoDisplayTask to ContentDisplay::VideoDisplay
- WebpageDisplayTask to ContentDisplay::WebpageDisplay
- SketchingTask to Drawing::SketchingWidget
- SessionManagementTask to Sessions::SessionManager
- CollaborativeDrawingTask to Drawing::CollaborativeDrawingWidget
- ConfigurationTask to Configuration::ConfigurationManager

5.2.2. ContentDisplay module

![Component diagram for the ContentDisplay module](image)

**Figure 5.2.2.a: Component diagram for the ContentDisplay module**

5.2.2.1. Processing narrative

The ContentDisplay module provides the necessary functionality for displaying the three primary kinds of visual content the IWBC supports: presentations (PresentationDisplay), videos (VideoDisplay) and webpages (WebpageDisplay). These display-related components provide the graphical basis for annotation, and they can access the external Screencasting component to start/stop the screencasting operation. The business logic for choosing which particular file will be displayed is also handled here, by the RecentlyUsed, FileSystemBrowser and GoogleDocsAccess components.
5.2.2.2. Interface description

IContentSelection: allows placing type restrictions on the selectable files and retrieving the selected file name.

IDisplayedContent: provides info the current status of the displayed content (current slide number for presentations, timeline location for videos, scroll position for webpages), used for recognizing the annotation context.

5.2.2.3. Processing detail

The ContentSelector component retrieves the selected filename from one of the components implementing the IContentSelection interface, and supplies the preferred ContentDisplay-derived display component (one of PresentationDisplay, VideoDisplay and WebpageDisplay) with the selected file. The display component then renders and displays the selected file, enabling the user to browse through the content (next/previous slide for presentations, playback control for videos, scrolling and link following for webpages) through its own user interface.

5.2.2.4. Dynamic behavior

- The ContentDisplay component sends the current status of the displayed content to Drawing::AnnotationWidget so it can display the relevant annotations [Sequence Diagram 11.1.3]
- When requested, the display component can access the external Screencasting component to start/stop the screencasting operation.

5.2.3. Drawing module

Figure 5.2.3.a: Component diagram for the Drawing module
5.2.3.1. Processing narrative

The Drawing module contains the components responsible for realizing the drawing operations which play a prime part in Watch & Touch’s functionality. The three components AnnotationWidget, SketchingWidget and CollaborativeDrawingWidget offer annotation, sketching (on blank pages) and collaborative drawing capabilities, respectively. The DrawingData component is responsible for storing the drawing data and importing/exporting into files when requested. The ScreenCapture component offers user-directed screen capturing capabilities, whose raster forms can be used as sketching backgrounds later on. The ContentMatcher component is responsible for recognizing the current state of the displayed content and fetch the relevant drawing data for annotations.

5.2.3.2. Interface description

IDisplayedContent: see section 5.2.2.2

All other inter-component links are associations, meaning that mutual access to both connected components’ public properties and functions are possible.

5.2.3.3. Processing detail

Drawing is performed on the user interface of the BaseDrawingWidget and derived components, and the drawing data is kept in the DrawingData component which can export or import this data as SVG images when requested. In addition to these basic drawing operations:

- The AnnotationWidget uses the ContentMatcher to identify the current status of the displayed content, and to retrieve the annotations which were previously made in this display state of the content, and display these annotations.
- The SketchingWidget can use the ScreenCapture component to retrieve a snapshot of the desired state of the screen, and use this as a background overlay to draw on [Sequence Diagram 11.1.4]
- The CollaborativeDrawingWidget updates the displayed drawing when other users in the collaboration session make changes to their own image.

5.2.3.4. Dynamic behavior

- The AnnotationWidget receives the status of the currently displayed content from the ContentDisplay::ContentDisplay component [Sequence Diagram 11.1.3]
- The CollaborativeDrawingWidget gets updated by the Sessions::Session component when change are made by other users in the session, and the Session is also notified when the user makes modifications to their own drawing [Sequence Diagram 11.1.5]
- The CollaborativeDrawingWidget negotiates the permission status of a student’s IWB access via their own CBC with the ClassroomSession component, allows modification on the current annotation state when permission is granted, and provides the current state of the displayed content and its annotation to the ClassroomSession [Sequence Diagram 11.1.6]
5.2.4. Sessions module

![Component diagram for the Sessions module]

**Figure 5.2.4.a: Component diagram for the Sessions module**

5.2.4.1. Processing narrative
The Sessions module includes the components related to the administration and functioning of the collaborative drawing sessions. The `MessageTransciever` component handles the sending and receiving of network messages which allow the communication of IWBC and CBCs through the calls of other Sessions components. The `SessionManager` is the façade component for the Sessions module and handles the creation and administration of sessions and users, thus allowing the IWBC to function as the collaboration hub. `SessionUsers` represent the participants of `Sessions`, and the `EmailSender` component is responsible for delivering e-mail copies of the collaborative drawing to the members of the session.

5.2.4.2. Interface description
No specific or constrained interfaces are defined; the `SessionManager` component is the façade component for the module and thus serves as the access interface.

5.2.4.3. Processing detail
When new users log in to the collaboration system and join or leave sessions the `SessionManager` is notified by the `MessageTransciever` component. The `SessionManager` then creates and modifies the corresponding `Session` or `SessionUser` objects as needed. When users enter or exit sessions or modify the collaborative drawing, the other CBCs in the session are notified by the `MessageTransciever` which is also capable of sending network messages. The `SnapshotRetriever` saves local copies of the current state of all collaborative drawings when requested, and `EmailSender` sends the local state of the collaborative drawing to all session members.
5.2.4.4. Dynamic behavior

- The ClassroomSession component uses the ContentDisplay::PresentationDisplay to view presentations as underlay during the classroom session [Sequence Diagram 11.1.6]
- The Drawing::CollaborativeDrawingWidget negotiates the permission status of a student’s IWB access via their own CBC with the ClassroomSession component, allows modification on the current annotation state when permission is granted, and provides the current state of the displayed content and its annotation to the ClassroomSession [Sequence Diagram 11.1.6]
- The Drawing::CollaborativeDrawingWidget gets updated by the Session component when change are made by other users in the session, and the Session is also notified when the user makes modifications to their own drawing [Sequence Diagram 11.1.5]

5.2.5. WiimoteInput module

![Component diagram for the WiimoteInput module](image)

**Figure 5.2.5.a: Component diagram for the WiimoteInput module**

5.2.5.1. Processing narrative

The WiimoteInput module forms the core of the IR-pen input functionality of the IWBC; it communicates with the Wiimote via the wiiuse library to receive the changes of the IR point position via the InputReceiver component, which forwards this to the InputCalibration component that generates screen coordinates for the IR point coordinates. The EventGenerator component generates the system mouse
events from the received coordinates, and if movements of multiple points are involved, the corresponding multi-touch gesture events as well. The **WiimoteManager** is the façade component for the module and provides access to the functionality of the other components, as well handling the initial connection operation to the Wiimote.

### 5.2.5.2. Interface description

No specific or constrained interfaces are defined; the **WiimoteManager** component is the façade component for the module and thus serves as the access interface.

### 5.2.5.3. Processing detail

Initially, the **WiimoteManager** uses the **wiiuse** library to find and connect to the Wiimote. Once the connection is established, the four-point calibration data as provided by the user interface is given to the **InputCalibration** component. When the IR source changes position, the **InputReceiver** gets the current position info via a callback function from **wiiuse**, and forwards the pure IR coordinates to the **InputCalibration** component, which translates them to screen coordinates. Finally, the **EventGenerator** creates mouse events based on the information received - including multi-touch gesture events if multiple points are detected to be forming a gesture.

### 5.2.5.4. Dynamic behavior

- The **InputCalibration** component saves and loads calibration data using the **Configuration::ConfigurationData** component.
- The **EventGenerator** passes multi-touch coordinates to **Configuration::GestureMapper**, which recognizes any multi-touch gestures and sends info about them back to the **EventGenerator**.
- The **WiimoteManager** sends battery status and calibration info to the **Configuration::ConfigurationManager** when requested.
- The **InputReceiver** and **WiimoteManager** use the **wiiuse** library’s API to access the Wiimote.
- see also: [Sequence Diagram 11.1.1] [Sequence Diagram 11.1.2]
5.2.6. Configuration module

5.2.6.1. Processing narrative
The Configuration module is responsible for keeping the user preferences regarding the functioning of the IWBC. The ConfigurationData module keeps all the system configuration, including the Wiimote calibration data, screencasting preferences, and the mapping of gestures and actions. The GestureMapper component handles the recognition of multi-touch gestures using the Wiimote input data. The ConfigurationManager component is the façade component for the module and provides access to the other components; enabling the changing of configuration through its user interface, as well as displaying information on the currently connected Wiimote.

5.2.6.2. Interface description
No specific or constrained interfaces are defined; the ConfigurationManager component is the façade component for the module and thus serves as the access interface.

5.2.6.3. Processing detail
Through its graphical user interface, The ConfigurationManager displays and modifies the current configuration information stored in the ConfigurationData. The GestureMapper keeps the current mapping of gestures and actions, and performs gesture recognition and identifying the mapped action when presented with multi-touch point data.
5.2.6.4. Dynamic behavior

- The ConfigurationData component receives updated Wiimote calibration data from WiimoteInput::InputCalibration when calibration is performed.
- The GestureMapper recognizes any multi-touch gestures and sends info about them to the WiimoteInput::EventGenerator when requested.
- The ConfigurationManager receives Wiimote calibration and battery info from WiimoteInput::WiimoteManager when requested.

6. User Interface Design

6.1. Overview of User Interface

Describe the functionality of the system from the user’s perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

6.1.1. User Interfaces for the IWBC

This section describes the graphical user interfaces offered by the Watch & Touch IWBC.

1. **Connect to Wiimote**: The user will be prompted to put Wiimote in the discoverable mode by simultaneously pressing 1+2 on the Wiimote, and will be informed about the progress of the connection.
6.1.1: Connect to Wiimote

2. **Calibration**: Four circles on the four corners of the screen will be displayed, each revealing the next one as it is touched. Once all four are touched, the user will be able to test the new calibration settings on the screen by scribbling around the page, and indicating whether to keep the settings or to repeat the calibration by a button press.
3. **IWBC Main Menu**: The main menu will be following a modified version of the desktop metaphor to allow the instructor to launch the various tasks offered by the IWBC. Several independent instances of the same task can be launched for certain types of tasks, while for others only a single instance will be allowed. Only a single task can be active at the same time, all tasks take up maximal area and cannot be resized. Switching between tasks and instances, or closing them will be also done via the main menu interface. A toolbar will provide access to the functionality listed below:

   a. Display Presentation - launch new instance of presentation viewer
   b. Display Webpage - launch new instance of webpage viewer
   c. Display Video - launch new instance of video viewer
   d. Sketch - launch new instance of sketching application
   e. Collaboration - open the multi-instance collaboration application
   f. Configuration - open the single-instance configuration application
   g. Exit - exit the IWBC

6.1.3.a: IWBC Main Menu – Annotation Menu On
4. **File Selection:** The IWBC applications that work with user content will require the selection of a file to be displayed (presentations, webpages, videos, sketches). This interface will provide three options to make this selection: the native file selection dialogue from the operating system, a list of recently used files in this category, and an option to select the file from the instructor’s Google Docs account. The user will be prompted to login to Google Docs if not already logged in. Cancelling the file selection operation to terminate the task instance and go back to the IWBC Main Menu is also possible.
5. **Display and Annotate Content:** The main interface for displaying presentations, webpages and videos, it will be offering the standard display control interface options (next/prev buttons for presentations, a browser bar for the webpages, playback control for videos) for each kind of content. If the loaded content had been annotated before, the annotation info will appear in the previously determined location (screen position for presentations and webpages, timeline position for videos). The user will also be able to annotate the content further by simply “writing” on the content when annotation is enabled from the annotation menu, or exit the interface to go back to the IWBC main menu. The CBC may ask to be granted permission to annotate the IWB, in such a case a notification will be displayed on the lower left hand corner. The user can choose to deny or grant this permission. Once the permission is granted, a small window will be visible on the lower left hand corner, which can be used to revoke the permission of the CBC anytime.
Abstract This paper introduces a novel framework for the design, modeling and control of a Micro Aerial Vehicle (MAV). The vehicle's conceptual design is based on biologically-inspired principles and emulates a dragonfly (Odonata–Anisoptera). We have taken inspiration from the flight mechanism features of the dragonfly and have developed indigenous designs in creating a novel version of a Flapping Wing MAV (FW-MAV). The MAV design incorporates a complex mechanical system, including a dedicated multi-layered, hybrid linear/non-linear controller to achieve short flight times and improved agility compared to other rotary wing MAVs and Vertical Take Off and Landing (VTOL) designs. The first MAV prototype, weighing a bare 30 g, can propel its own 15 g body and has a maximum velocity of about twice the weight. The MAV features state-of-the-art sensing and instrumentation payload, which includes integrated high-power on-board processors, 6DoF inertial sensors, 3DoF compasses, GPS, embedded camera and long-range telemetry capability. A 3-layer control mechanism has been developed to harness the dynamics and maintain complete navigational control of the MAV. The inner-layer is composed of a ‘quad hybrid-energy controller’ and two higher layers are at present, implementing a linear controller; the latter will be replaced eventually with a dynamic adaptive non-linear controller. The advantages of the proposed design compared to other similar ones include higher energy efficiency.

Fig. 6 Wing sweep: range of motion

2. The value of $C_D$ has been chosen to be equal to 0.01, to correspond to a flat plate parallel to the flow of air.
Fig. 6 Wing sweep: range of motion

6.1.5.c: IWBC Presentation – Page Slider

6.1.5.d: IWBC Web
6. **Annotation Menu**: Contains context-dependent annotation facilities: enabling/disabling annotation, changing pen style and color, eraser, exporting the annotated content, clearing all annotations and starting / stopping the screencast operation. The menu can be reached by performing the Open Context Menu multi-touch gesture determined in the Configuration section, or by long-pressing the IR pen (which is equivalent to a right click).
7. **Sketch**: Provides general drawing facilities similar to those found in painting applications. A pre-existing image can be loaded into the sketching area, and created sketches can be saved as images. Alternatively, the user can ask to work on a capture of the screen - in this case the IWBC will be minimized until the user presses the “Print Screen” key, which will capture the current screen image and bring it to the Sketch application as an underlay to be worked on.

8. **Sketching Menu**: Contains a variety of sketching-related tools: changing pen style and color, eraser, drawing elementary geometric shapes, saving/loading the image and capturing the screen. The menu can be reached by performing the Open Context Menu multi-touch gesture determined in the Configuration section, or by long-pressing the IR pen (which is equivalent to a right click).
9. **Session Manager**: The portal for collaboration on the IWBC side, this interface provides the user with a list of currently existing collaborative drawing sessions. The instructor will be able to view the names of participants in each list. The user will also be able to create a new session by specifying a name and a password, or join an existing session (without providing the session password - this is a privilege of the IWBC user). Another feature accessible from this interface is the option to gather snapshots of all current collaborative drawings.

![Session Manager Interface](image)
**10. Collaborative Drawing:** In this interface, the user will be able to create drawings together with the other members of the collaborative drawing session. Each user will be able to modify the drawing and see the modifications done by others as soon as the modifications are made, see the list of session members and change the currently used drawing tools, export or e-mail the current state of the drawing, or import a previously made drawing into the current session via a toolbar. Making local (not collaborative, not accessible to others) drawings is possible on a separate tab of the interface.

*6.1.10: IWBC - Collaborative Drawing*
11. **Configuration**: This interface will allow the user to change the mapping of predetermined multi-touch gestures to actions, re-calibrate the Wiimote and see the current battery level and calibration status of the Wiimote.

---

**Settings**

<table>
<thead>
<tr>
<th>Predefined Gestures</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-Finger Swipe</td>
<td>Open annotation/sketching/drawing menu</td>
</tr>
<tr>
<td>Pinch In</td>
<td>Maximize current task</td>
</tr>
<tr>
<td>Pinch Out</td>
<td>Unmaximize current task</td>
</tr>
<tr>
<td>Tap &amp; Hold</td>
<td>Hide main menu bar</td>
</tr>
</tbody>
</table>

---

*6.1.11: IWBC Configuration*
6.1.2 User Interfaces for the Collaborativity Client:

1. **Login:** To be able to access the collaborative working system, the user must provide his name and surname, (optionally e-mail address and student number). This simple interface allows the user to provide these credentials and login to the system.

   ![CBC Login](image)

   *6.1.2.1: CBC Login*

2. **CBC Main Menu:** The portal for collaboration on the CBC side, this interface provides the user with a list of currently existing collaborative drawing sessions. The user will also be able to create a new session by specifying a name and a password, or join an existing session by providing the session password. On the CBC side, the user is able to join the special Classroom Session directly from this interface (no password required).
6.1.2.2.a: CBC Main Menu

6.1.2.2.b: CBC Main Menu, Hide On
3. **Collaborative Drawing**: identical to the IWBC Collaborative Drawing interface, see section 3.1.1.10 for description

---

6.1.2.3: CBC Collaborative Drawing
4. **Classroom Session**: This interface will be identical to the Collaborative Drawing interface in appearance, with some extra features. The background of the drawing will always reflect the current content displayed by the IWB, and the drawings that the CBC user makes on top will not be reflected on the IWB unless the CBC asks for special permission via a button on the toolbar.

6.1.2.4.a: CBC Classroom Session

6.1.2.4.b: Asking for the Chalk

*Chalk is asked by pressing ‘chalk’ button[1].
6.2. Screen Images

6.2.a: IWBC – Annotation Menu, File Operations

\[ \begin{align*}
\dot{q} &= -\dot{\theta}_1 \\
\dot{\theta}_1 &= \theta - \theta_1 \\
\dot{\theta}_2 &= \theta_2 - \theta_1 \\
\end{align*} \] 

where, \( \theta = \frac{\pi}{2} \) and \( \theta_2 = \frac{\pi}{4} \), \( x \) is the distance between the pivot 1 and the position of the common (P1) and \( \varphi \) is the angle made by \( x \) and \( \varphi \). Further explanation and explained in chapter 4.
6.3. Screen Objects and Actions

1. Annotation Menu

- a. Zoom-In menu
- b. Zoom-Out menu
- c. Change location of the menu

2. Screen Casting Indicator

3.1.3: Session Manager Menu

If screen casting is started, ‘a’ is shown on the bottom left side of the screen. It turns to a’ if screen casting is stopped.

3. Multimedia Control Panel

- a. Play - pause button
- b. Go to previous - next annotation
- c. Close recent annotation
- d. Annotation indicator on timeline
- e. Timeline indicator
7. Detailed Design

This section contains the internal details of the design entities whose overview was presented in the Architecture section. These details include attribute descriptions for identification, processing and data. All necessary details that will be needed by the developers for implementation can be found here.

7.1. Detailed Module Description

This section contains class diagrams and function textual descriptions of the classes belonging to the modules whose general description was provided in section 5.2.

Certain formatting conventions are used in the textual descriptions to refer to design details:

- **Class**
  - **ModuleName::Class**
  - **classMemberFunction**
  - **classMemberProperty**

*Note: for the sake of clarity, classes which do not belong to the described module but have first-level associations with one of the classes in the module are also shown on the class diagrams. The textual descriptions do not include function descriptions for these classes - their descriptions can be found in the section corresponding to their own module.*
7.1.1.1. The Tasks System

Watch and Touch is capable of executing a number of different tasks (sketching, presentation display, collaborative drawing...), each of which is represented as a *Task object. Except for the ConfigurationTask and the SessionManagementTask which have only one instance, multiple instances of these tasks can be created and be active at the same time. Each task has a unique identifier taskID, and serves as a container to a primaryWidget which carries out the user interface display and the actual functionality of the task. The task is responsible for emitting the taskStateChanged signal when a change occurs in the task state (see section 4.1.1.14 for possible task states).
The TaskManager class is responsible for managing these task instances, both logically and visually. The Watch & Touch main menu is presented by the showTaskManagerUI function, which forms the user interface for task management. The exitApplication function can be used to terminate Watch & Touch and all task instances safely. In terms of business logic of the task management, the class defines the following functions:

- **createTaskInstance** - creates a new task instance of the given type (see TOOD for possible task types). A task of the given type is created, added to the list of running tasks and returned. For single-instance tasks, this function returns the existing instance for the task.
- **getTaskByID** - returns the Task object corresponding to the given task identifier, null otherwise.
- **terminateTask** - terminates the task corresponding to the given task identifier. No action is carried out if the task with the given ID does not exist.
- **getInstanceCount** - returns the number of task instances of the given task type.
7.1.2. ContentDisplay module

7.1.2.1. The content selection subsystem

The content display process starts with the user specifying which particular content they would like to display and annotate. The content selection subsystem is responsible for handling the reception of the user’s choice. Three different sources are available to choose from: the local filesystem (FileSystemBrowser), Google docs (GoogleDocsAccess) and the list of recently used files (RecentlyUsed). These components all implement the IContentSelection interface, which includes functions for displaying the selection UI, and emitting the contentSelected signal when the user confirms their selection. The ContentSelector acts as a container for these three options, and when the selection is confirmed it recognizes the type of the content, creates the relevant type of ContentDisplay widget and passes the filename to it.
7.1.2.2. Content display basics

The QWidget-derived ContentDisplay forms the base of all three types of content display widgets, and includes their common functionality:

- `selectContent` - opens the content given its location, which can be a local file path or URL
- `getContentSize` - returns the visual size of the content in pixels
- `getContentMD5` - returns the MD5 hash of the content file, used for accurately identifying the content for annotation purposes (see section 4.1.2.1 for details)

7.1.2.3. Displaying presentation content

The PresentationDisplayWidget is responsible for displaying presentations, and offers basic slide flow control with the `gotoPrevSlide`, `gotoNextSlide` and `gotoSlide` functions which are connected to the user interface. The slides that are being viewed are always QImages, thus a conversion process actually takes place behind the scenes to generate these slide images. To open a presentation, the filename is first passed to PresentationLoader, which identifies the file type of the presentation (PDF, PPT, ODF). Non-PDF presentations are first converted to the PDF format via the external command line converters. Once a PDF file is obtained, the PDFLoader class is given the PDF file name to start converting the pages to images. Working as a separate thread, the PDFLoader generates the page images using libpoppler. A pageReady signal is emitted to the PresentationLoader as each page is converted, whose `getPageImage` function can be used to access the generated page images. The generated page images will be kept in the PDF image cache folder under Watch & Touch, so that the conversion does not have to take place again when the same PDF is opened the next time.

For presentations, the `contextChanged` signal for the IDisplayedContent interface is emitted when the displayed slide number changes, and contains the new slide number and the content hash data.

7.1.2.4. Displaying video content

The Qt multimedia framework Phonon is utilized for video playback through the VideoDisplayWidget. The QWidget-derived Phonon::VideoPlayer class offers all the necessary functions for displaying video, including the timeline control functions `play`, `pause`, `seek` and `stop`.

Since the annotation context information for videos is defined as the timeline position of the video with 1-second resolution, the internal timer generates a timerEvent every second while the video is playing, and the `contextChanged` signal for the IDisplayedContent interface is emitted with the current timeline position and the content hash data.

7.1.2.5. Displaying webpage content

The WebView class from Qt’s WebKit module will be providing the webpage display functionality, which can handle display of local HTML files as well as those over the Internet. The `reload`, `back` and `forward` control functions are provided and can be accessed from the user interface for controlling the browsing.

When the user clicks on a link on the webpage (internal anchor or external), the `contextChanged` signal for the IDisplayedContent interface is emitted with the current URL anchor and the content hash data.
### 7.1.3. Drawing module

The **BaseDrawingWidget** forms the base of all the other drawing-related widgets in Watch & Touch. The drawing system makes use of the Qt Graphics View framework; the drawn items (lines, circles, etc.) are **QGraphicsItems** wrapped inside a **DrawingAction**, which are inserted into the **DrawingData** derived from **QGraphicsScene**. The **QGraphicsView-derived** **BaseDrawingWidget** provides a view of the **DrawingData** and represents the drawing at its highest level. Each drawing may have multiple pages associated with it, only the active page (**setPageNo**, **getPageNo**) is displayed in the widget.

---

**ScreenCapture**
- capturedImage : QImage
  + captureScreen()
  - keyBoardEventMonitor()
  + getCapturedImage() : QImage

**SketchingWidget**
+ showContextMenu()
+ saveSketch(fileName : string)
+ loadSketch(fileName : string)

**AnnotationWidget**
- currentPage(): int
  + showContextMenu()
  + clearAllAnnotations()
  << QSLOTS >> +contextChanged(context : string)
  << QSLOTS >> +contextChanged(context : string)

**BaseDrawingWidget : QGraphicsView**
- pages : DrawingData[]
- pageNo : int
- drawingPen : QPen
- drawingBrush : QBrush
- mode : DrawingMode
- backgroundBrush : QBrush
  + setPageNo(pageNo : int)
  + getpageNum() : int
  + showContextMenu()
  + operation()
  + setDrawingPen(pen : QPen)
  + getDrawingPen() : QPen
  + setDrawingBrush(brush : QBrush)
  + getDrawingBrush() : QBrush
  + getModel() : DrawingMode
  + setModel(mode : DrawingMode)
  + expoint(implicit : string, format : string)
  + setCurrentPage()
  + clearAllPages()
  + getBackgroundBrush() : QBrush
  + setBackgroundBrush() : QBrush
  + saveDrawing(fileName : string)
  + loadDrawing(fileName : string)

**CollaborativeDrawingWidget**
+ showContextMenu()
+ saveDrawing(fileName : string)
+ loadDrawing(fileName : string)
<< QSlot >> +updateDrawing(update : DrawingAction)
<< QSignal >> +drawingUpdated(update : DrawingAction)

**ContentDisplay : QWidget**
- contentLocation : string
- contentSize : QSize
- contentMD5 : string
  + selectContent(contentLocation : string) : boolean
  + getContentSize() : QSize
  + getContentMD5() : string
  + generateMD5(contentLocation : string)

**DrawingData : QGraphicsScene**
- undoStack : QUndoStack
- metaData : QHash
  + saveSVG(fileName : string)
  + loadSVG(fileName : string) : boolean
  + clearContent()
  + setMetaData(field : string, value : string)
  + getMetaData(fileName : string) : string

**DrawingAction : QUndoCommand**
- myGraphicsItem : QGraphicsItem*
  - description : string
  + redo()
  + undo()
  + getDescription() : string
  + setDescription(description : string)

**Session**
- userList : SessionUser[]
- sessionID : int
  + addUser(username : string)
  + removeUser(username : string)
  + getSessionID() : int
  + setSessionID(sessionID : int)
<< QSlot >> +updateDrawing(update : DrawingAction)
<< QSlot >> +drawingUpdated(update : DrawingAction)
The style and properties of the drawing currently made on the BaseDrawingWidget is influenced by the current drawingPen (line color, thickness, style), drawingBrush (filling color, thickness, style) and mode (line, eraser, circle...). The background color of the drawing is determined by the backgroundBrush.

The DrawingAction class itself is derived from QUndoCommand to make use of the Qt Undo/Redo framework - this way, lines or shapes made on the drawing are each represented as a different object which can be added to or removed from the drawing. This incremental drawing representation also allows for easier implementation of collaborative drawings.

Drawings can have associated textual metadata in them, which are manipulated through setMetaData and getMetaData.

The drawing can be made into persistent SVG form using saveDrawing, or loaded from a file with loadDrawing. These operations invoke respectively the saveSVG and loadSVG operations inside the relevant DrawingData. The drawing can be saved as PNG, JPEG or any other binary raster format supported by Qt using the export function.

7.1.3.2. Annotation system

In Watch & Touch, annotations are defined as user-made drawings which are associated with visual content (presentations, webpages, videos) and are handled by the AnnotationWidget class. The visual content and the drawing exist in separate widgets; thus the AnnotationWidget is placed on top of the relevant ContentDisplay widget. No special action needs to be taken by the user to load previously made annotations for the content; when a new content file is loaded, or the context of the content file changes (e.g. move to the next slide in a presentation) the AnnotationWidget is notified by the IDisplayedContent interface signal contextChanged, and the ContentMatcher class fetches the relevant annotation filename as described in section 4.1.2.1.

7.1.3.3. Collaborative drawing system

The CollaborativeDrawingWidget is essentially a widget which receives additional drawing commands from other users via the Sessions module as described in section 7.1.4. Updates on drawings by users are distributed to the whole session as a DrawingAction. When the user makes changes on his/her own drawing, the updateDrawing signal is emitted, and the drawingUpdated signal is received when changes are made by other users to the drawing.

7.1.3.4. Sketching

Sketching in Watch & Touch is utilizing the drawing functionality at its most basic level, and the SketchingWidget is used to perform this. The only additional capability offered on top of BaseDrawingWidget is the ability to take screenshots via the ScreenCapture class and scribble on them.

Once a screenshot is requested, the ScreenCapture will start monitoring keyboard events via its keyboardEventMonitor, and capture the screen image when it detects a PrintScreen keypress. The captured screen QImage will be returned to the SketchingWidget, and displayed as the drawing underlay if accepted by the user.
7.1.4. Sessions module

The session system and collaborative drawing require a method of sending and receiving data among different classes located on different physical machines, and in this section we will be describing how this mechanism works in Watch & Touch.

- The overall picture is actually a classical client/server model, where the IWBC is acting as the server and the CBCs are acting as clients (actually, the IWBC also acts as a client itself, since the instructor can also take part in the collaborative drawing activity).
The IWBC acts as the main data storage (the status of each collaborative drawing, the objects representing sessions and users are all kept in the IWBC machine) and the communication hub (CBCs do not talk to each other directly, all communication passes through the IWBC).

The classes inside the Sessions module represent the functional partitioning of collaborative drawing-related entities, such as Session which represents a single session and SessionUser which represents a collaborative drawing participant. Objects formed from these classes are present both in the IWBC and the CBCs. The IWBC has objects for every single collaborative drawing entity, while the CBCs only have the objects that concern them (i.e. the Session of which they are a member and the SessionUsers in that session). The objects corresponding to the same entity are kept in sync through messages.

These messages are sent between the SessionManagers on the IWBC and the CBCs - other classes in the Sessions module do not send messages themselves, but call the relevant functions from the SessionManager instead. The SessionManager is thus responsible for sending and receiving the necessary messages to the right places to keep the objects in sync.

The operation mode of the SessionManager is set by the operationMode property: if it is set to IWBC then the SessionManager acts as a server, and if it is set to CBC it acts as a client. See the end of section 7.1.4.2 for differences on how server and client SessionManagers operate.

### 7.1.4.2. Defined Messages for Collaboration

<table>
<thead>
<tr>
<th>ID</th>
<th>Direction</th>
<th>Message Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserLogin</td>
<td>CBC -&gt; IWBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>UserLoginResponse</td>
<td>IWBC -&gt; CBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>UserLogout</td>
<td>CBC -&gt; IWBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>UserLogoutResponse</td>
<td>IWBC -&gt; CBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>SessionList</td>
<td>CBC -&gt; IWBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>SessionListResponse</td>
<td>IWBC -&gt; CBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>JoinSession</td>
<td>CBC -&gt; IWBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>JoinSessionResponse</td>
<td>IWBC -&gt; CBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>LeaveSession</td>
<td>CBC -&gt; IWBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>LeaveSessionResponse</td>
<td>IWBC -&gt; CBC</td>
<td>userName:string</td>
</tr>
<tr>
<td>UpdateDrawing</td>
<td>CBC -&gt; IWBC</td>
<td>userName:string</td>
</tr>
</tbody>
</table>
The manner of creation of messages depends on the server or client behaviour of the **SessionManager**:

- CBC-side messages always include the CBC user’s username for source identification, and are always sent to the IWBC.
- IWBC-side messages always include the target user’s username, so that the **MessageTransciever** can deliver the message to the right user.

7.1.4.3. Data Communication via MessageTransciever

To abstract the remaining classes from the details of network communication, the **MessageTransciever** class is used. Although the class is defined in the same manner on the CBC and IWBC, it is used differently:

- The CBC-side **MessageTransciever** uses the `connectToServer` function to connect to the IWBC-side **MessageTransciever**, which runs `listenForConnections` instead.
- Once the IWBC gets the connection request from a CBC, the connection is established and a pair of QTcpSockets are created.
- The CBC **MessageTransciever** stores the socket as the `serverSocket`, and sends the UserLogin message.
- Upon receiving the UserLogin message from the newly created message, the IWBC **MessageTransciever** stores the socket in the `clientSockets` hash, with the username as the key.
- Messages are exchanged over the established connection between the two transcievers.

7.1.5. WiimoteInput and Configuration modules
7.1.5.1. Receiving input from the Wiimote

The reception of IR point locations from the Wiimote and their translations into mouse events or gestures takes place inside the WiimoteInput module, as described in section 5.2.5. When Watch & Touch is first started, the `connectToWiimote` function from the `WiimoteManager` is called, and the user interface 6.1.1.1 is displayed, prompting the user to put the Wii in discoverable mode. In the meanwhile, the `InputReceiver`'s `connectToWiimote` function is invoked, which calls `wiiuse_init` to initialize the library, `wiiuse_find` to discover Wiimotes and `wiiuse_connect` to actually establish the connection to the Wiimote. Once the connection is established, the `inputEventLoop` starts running on a separate thread and receiving the IR point coordinates by calling `wiiuse_poll`. The `InputCalibration`'s `recalibrate` function is then called to display the user interface for calibration (6.1.1.2) and receive the four `calibrationPoints`. Once the calibration is complete, the calibration interface disappears and the `InputCalibration` class starts calibrating the received IR point data according to the received calibration points, then sends these points to `EventGenerator`. If only a single IR point is present, the `EventGenerator` generates a mouse event through the `generateMouseEvent` function. If multiple points are present, the `GestureMapper`'s `recognizeGesture` function is called to identify and possible gestures, which are generated as the corresponding actions (section 4.1.1.12) through the `generateGestureEvent` function in case any of the recognizable gestures have occured (section 4.1.1.11).
7.1.5.2. Configuration
The configuration data is kept in persistent form by the `ConfigurationData` class, whose `saveConfiguration` and `loadConfiguration` functions can be used to store or load the configuration data in XML form (see section 4.1.1.9). The ConfigurationManager class is responsible for displaying the user interface for modifying the configuration through the `showConfigurationUI` function, as well as provide access to the existing configuration through the `getConfigurationValue` and the `setConfigurationValue` functions.

7.1.5.3. Screencasting
The screencasting operation allows a recording of all screen activity taking place on the IWBC display. The Screencasting component is accessed via the `ConfigurationManager` and its functions `startScreencasting` and `stopScreencasting` which are triggered via the screencasting button on the main interface (see section 6.3.3) allow for controlling the screencasting operation.
7.2. Detailed design for the Collaboration Client

```
<<singleton>>
ClassroomSession
- WBCAccess : string
- sessionID : int = 1
+ openPresentation()
+ getWBCAccess(): string
+ setWBCAccess(user Name : string)

Session
-userList : SessionUser[]
-sessionID : int
+ addUser(user Name : string)
+ removeUser(user Name : string)
<<QSignal>> + updateDrawing(update : DrawingAction)
<<QSlot>> + drawingUpdated(update : DrawingAction)
+ getSessionID(): int
+ setSessionID(sessionID : int)

<<singleton>>
SessionManager
+ activeSessions : Session[]
+ activeUsers : SessionUser[]
+ createSession(sessionName : string, password : string, founderName : string): boolean
+ destroySession(sessionName : string)
+ joinSession(sessionName : string, password : string, userName : string): boolean
+ leaveSession(sessionName : string, userName : string)
+ getUserName(): SessionUser[]
+ getUserList(): SessionUser[]
+ setUserName(username : string, eMailAddress : string): boolean
+ sendLoginPassword(username : string)
+ getSession(sessionID : int): Session
+ getUser(userName : string, sessionID : int)
+ addDrawing(sessionName : string, update : DrawingAction)
+ requestPermission(requesterName : string, requesteeName : string, action : string)
+ sendPermissionRequest(requesterName : string, requesteeName : string)
+ updateDrawing(sessionName : string, update : DrawingAction)
+ getWBCAccess(identifier : string)

SessionUser
- userName : string
- eMailAddress : string
+ getUserName(): string
+ setUserName(userName : string)
+ getEMailAddress(): string
+ setEMailAddress(eMailAddress : string)

<<singleton>>
TaskManager
+ activeTasks : Task[]
+ getTask(taskID : int): Task
+ createTask(taskname : TaskType, Task): Task
+ deleteTask(taskID : int)
<<QSignal>> + taskStateChanged(newState : TaskState)
+ getRemainingCount(taskType : TaskType): int
+ exit(): void
+ showTaskManagerUI()
As was described in section 5.1.2, the collaboration client is essentially formed from a subset of IWBC components (which actually also serves as a collaboration client itself, since the instructor is also able to take part in collaborative drawing activity), thus the detailed class descriptions given in section 7.1 also apply here. The class diagram provided above illustrates the connections between the classes forming the CBC.

Although the classes used are the same, the functionality utilized in these classes is more limited compared to the IWBC. These points of limitation and/or different behaviour are as follows:

- The TaskManager only supports two types of tasks: CollaborativeDrawingTask and SessionManagementTask
- The displayed user interfaces are not the same as IWBC user interfaces - they are described in section 6.1.2
- The SessionManager acts in the CBC mode (see section 7.1.4.2)
- The MessageTransciever uses the connectToServer function and obtains the serverSocket instead of keeping clientSockets

8. Libraries and Tools

Watch & Touch will make use of several existing software libraries and tools, both during development to achieve a faster production cycle and during runtime to obtain extra functionality. These libraries and tools are described in the following subsections.

8.1 Qt

8.1.1. Description

Qt is a cross-platform application framework that is widely used for developing application software with graphical user interface (GUI) (in which case Qt is referred to as a widget toolkit when used as such), and also used for developing non-GUI programs such as command-line tools and consoles for servers. Qt is most notably used in Autodesk, Google Earth, KDE, Adobe Photoshop Album, the European Space Agency, OPIE, Skype, VLC media player, Samsung, Philips, Panasonic and VirtualBox. It is produced by Nokia’s Qt Development Frameworks division, which came into being after Nokia’s acquisition of the Norwegian company TrollTech, the original producer of Qt. Qt uses standard C++ but makes extensive use of a special code generator (called the Meta Object Compiler, or moc) together with several macros to enrich the language. Qt can also be used in several other programming languages via language bindings. It runs on all major platforms and has extensive internationalization support. Non-GUI features include SQL database access, XML parsing, thread management, network support, and a unified cross-platform API for file handling. Distributed under the terms of the GNU Lesser General Public License (among
others), Qt is free and open source software. All editions support a wide range of compilers, including the GCC C++ compiler and the Visual Studio suite.

8.1.2. Usage in Watch & Touch

By the nature of its primary feature set (annotation, multitouch gesture manipulation, collaborative drawing) Watch & Touch is quite user interface centric. This heavy emphasis on graphical user interfaces makes their ease of construction a primary factor in design. Its feature-rich and diverse library of UI elements (including a multimedia framework for viewing videos, an HTML renderer for webpages and SVG support), multitouch gesture capabilities and the Qt Creator IDE make Qt an excellent choice of framework for Watch & Touch. Another important factor is the framework’s cross-platform support; with Qt, building applications for a range of operating systems is often a simple matter of recompiling.

8.2 wiiuse

8.2.1. Description

Wiiuse is a library written in C that connects with several Nintendo Wii remotes. Supports motion sensing, IR tracking, nunchuk, classic controller, and the Guitar Hero 3 controller. Single threaded and nonblocking makes a light weight and clean API.

8.2.2. Usage in Watch & Touch

The wiiuse library is used in Watch & Touch as a layer of abstraction between the IWBC and the Wiimote; it handles the Bluetooth communication between the Wiimote and the instructor’s machine and exposes a number of methods for accessing the Wiimote’s functionality without going into the details of the communication protocol.

8.3 Google Documents API

8.3.1. Description

The Google Documents List API allows client applications to programmatically access and manipulate user data stored with Google Documents. Here are some of the things you can do with the API:

- Discovery: Retrieve documents that match specific keywords, categories, or metadata.
- Download: export documents in common formats such as pdf, rtf, doc, xls, ppt, and more.
- Sharing (ACLs): Modify the sharing permissions of documents and folders. The API allows sharing to individuals, group emails, or across an entire Google Apps domain.
- Create/upload/copy documents: Create online backups of local word processor documents, spreadsheets, presentations, and PDFs.
- Revisions: Review, download, or publish a document’s complete revision history.
- File documents: Create folders and move documents/folders in and out of folders.
• Spreadsheets: While the Documents List API can be used to create and retrieve a list of Google Spreadsheets, it cannot be used to modify the data within a Spreadsheet. For that, you can use the Google Spreadsheets API.

8.3.2. Usage in Watch & Touch
In the Watch & Touch IWBC, the Google Documents List API is used to provide an alternative input source for presentations - instead of using local files, the user can provide Google Account login details to access the Google Docs presentations stored in this account. The ClientLogin authorization method is used to gain access to the account, followed by API calls that retrieve the list of presentations and download a given presentation in the PDF format. The details of the API are abstracted from the rest of the application by the GoogleDocsAccess component, which provides direct methods for logging in, getting the list of presentations and downloading a given presentation as PDF.

8.4 Screencasting tools

8.4.1. Description
A screencast is a digital recording of computer screen output, also known as a video screen capture, often containing audio narration. The term screencast compares with the related term screenshot; whereas a screenshot is a picture of a computer screen, a screencast is essentially a movie of the changes over time that a user sees on a computer screen, enhanced with audio narration. Screencasting tools are the software tools that allow the recording of such screen activity videos.

8.4.2. Usage in Watch & Touch
A variety of screencasting applications[5] exist; however, keeping in mind that Watch & Touch aims to be open-source and cross-platform compatible, programs which fulfill these two constraints would be more fitting, so the two suitable candidates have been determined as VLC and FFmpeg. The user is able to specify which of these tools will be used during the screencasting operation in the IWBC Configuration screen.

8.5. Poppler

8.5.1. Description
Poppler (or libpoppler) is a free software library used to render PDF documents. It is used by the PDF viewers of the open source GNOME and KDE desktop environments, and its development is supported by freedesktop.org. The project was started by Kristian Høgsberg with two goals in mind: To provide PDF rendering functionality as a shared library, in order to centralize maintenance effort, and to go beyond the goals of Xpdf, and integrate with functionality provided by modern operating systems. Poppler itself is a fork of the Xpdf-3.0 PDF viewer developed by Derek Noonburg of Glyph and Cog, LLC.
8.5.2. Usage in Watch & Touch

Watch & Touch uses the libpoppler Qt4 interface, which allows rendering of PDF files directly at the desired resolution (as well as numerous other operations) from inside Qt to display PDF content. The following code segment illustrates how easily this can be accomplished:

```cpp
Poppler::Document* document = Poppler::Document::load(filename);
Poppler::Page* pdfPage = document->page(pageNumber);

// Generate a QImage of the rendered page
QImage image = pdfPage->renderToImage(xres, yres, x, y, width, height);

// ... use image ...
```
9. Time Planning (Gannt Chart)

9.1. Term 1 Gannt Chart

<table>
<thead>
<tr>
<th>Task</th>
<th>Start</th>
<th>End</th>
<th>Duration (Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Project Management</td>
<td>10th Oct</td>
<td>10th Nov</td>
<td>04 - Oct - 10</td>
</tr>
<tr>
<td>1.1 Reqirement Report</td>
<td>11th Oct</td>
<td>11th Nov</td>
<td>11 - Oct - 10</td>
</tr>
<tr>
<td>1.2 Analysis Report</td>
<td>18th Oct</td>
<td>18th Nov</td>
<td>18 - Oct - 10</td>
</tr>
<tr>
<td>1.4 Final Design Report</td>
<td>01st Nov</td>
<td>01st Dec</td>
<td>01 - Nov - 10</td>
</tr>
<tr>
<td>1.5 Final Design Report</td>
<td>08th Nov</td>
<td>08th Dec</td>
<td>08 - Nov - 10</td>
</tr>
<tr>
<td>1.6 Final Design Report</td>
<td>15th Nov</td>
<td>15th Dec</td>
<td>15 - Nov - 10</td>
</tr>
<tr>
<td>1.7 Final Design Report</td>
<td>22nd Nov</td>
<td>22nd Dec</td>
<td>22 - Nov - 10</td>
</tr>
<tr>
<td>1.8 Final Design Report</td>
<td>29th Nov</td>
<td>29th Dec</td>
<td>29 - Nov - 10</td>
</tr>
<tr>
<td>1.9 Final Design Report</td>
<td>06th Dec</td>
<td>06th Jan</td>
<td>06 - Dec - 10</td>
</tr>
<tr>
<td>1.10 Final Design Report</td>
<td>13th Dec</td>
<td>13th Jan</td>
<td>13 - Dec - 10</td>
</tr>
<tr>
<td>1.11 Final Design Report</td>
<td>20th Dec</td>
<td>20th Jan</td>
<td>20 - Dec - 10</td>
</tr>
<tr>
<td>1.12 Final Design Report</td>
<td>27th Dec</td>
<td>27th Jan</td>
<td>27 - Dec - 10</td>
</tr>
<tr>
<td>1.13 Final Design Report</td>
<td>03rd Jan</td>
<td>10th Jan</td>
<td>03 - Janv - 11</td>
</tr>
<tr>
<td>1.14 Final Design Report</td>
<td>10th Jan</td>
<td>17th Jan</td>
<td>10 - Janv - 11</td>
</tr>
<tr>
<td>1.15 Final Design Report</td>
<td>17th Jan</td>
<td>24th Jan</td>
<td>17 - Janv - 11</td>
</tr>
<tr>
<td>1.16 Final Design Report</td>
<td>24th Jan</td>
<td>31st Jan</td>
<td>24 - Janv - 11</td>
</tr>
</tbody>
</table>
10. Conclusion

This document states the design level approach taken by DialecTech team for the Watch & Touch project. In this document, a fair amount of elaboration has been done on the project scenario pointing out most of the important details. The goals for the final product has become more apparent as the scenario and the desired user interface is visually explained. Additionally, this document is the first document that explains somewhat deep technical details. The architecture of the system is discussed with an overview, and illustrated with component diagrams. Further information on the technical design is given with detailed explanations of the modules which are supported with and sequence diagrams. Finally, the progress made by the project team has been summarized.

11. Appendix

11.1. Sequence Diagrams

11.1.1. Wiimote Initialization sequence diagram
11.1.2. Wiimote Calibration sequence diagram

1. request recalibration
2: getNewCalibration()

2.1: getNewCalibration()
3: four calibration points

4: getNewCalibration()

5: prompt to test settings
6: accept settings

6.1: setCalibrationData()

11.1.3. Content Display and Annotation sequence diagram

1: request content display
2: getContentSelection()

2.1: getContentSelection()

3: present selection options
4: select content

5: getContentSelection()

6: content is displayed
7: browse content

8: getMatchingAnnotation()

9: recognizeContext()

10: getCurrentContent()

11: getCurrentContent()

12: recognizeContext()

13: getMatchingAnnotation()

14: annotation (if exists) is loaded and displayed

15: modify annotations
11.1.4. Sketching sequence diagram

User

1: new sketch
2: request screenshot
5: scribble

SketchingWidget

1.1: clear()
2.1: getScreenCapture()
4.1: getScreenCapture()
5.1: modify

DrawingData

3: prompt for pressing PrintScreen key

ScreenCapture

4: press PrintScreen
2.2: setBackgroundImage()
11.1.5. Collaborative Drawing overview sequence diagram

Collaboration Client #1

1: login

1.1: loginMessage

1.1.1: loginUser()

1.2: login is OK

1.2: create new session

2: create new session

2.1: createSessionMessage

2.1.1: createSession()

2.2: session created

Message Transceiver (CSC)

3: login

3.1: loginMessage

3.1.1: loginUser()

3.2: login is OK

3.2: create new session

3.2.1: createSessionMessage

3.2.1.1: createSession()

3.2.2: session created

Message Transceiver (JWBC)

4: update drawing

4.1: drawingMessage

4.1.1: updateDrawing()

4.1.1.1: drawingUpdated()

Session Manager

4.1.1.1.1: update drawing
11.1.6. Classroom Session sequence diagram

 Students

 Collaborative Drawing Widget (CBC Side)

 Collaborative Drawing Widget (IWBC Side)

 Classroom Session (IWBC Side)

 Instructor

 1.1: content updated

 2.1: request IWBC access

 2.1.1: permission request

 2.1.2: permission granted

 3: grant permission

 4: sortable

 4.1: content updated

 1: annotation on presentation

 2.1.1.1: show permission request

 2.1.1.2: show permission request
11.2. Detailed Use-Case Diagrams

11.2.1. Calibration use-case diagram

11.2.2. Log in and Log out use-case diagrams
11.2.3. Create Student Account and Instructor Account use-case diagrams

- Provide account creation screen
  - Instructor
  - System
- Enter and submit username and password
  - Instructor
  - System
- Enter e-mail address (optional)
  - Instructor
  - System
- Create instructor account
  - Instructor
  - System
- Provide account creation screen
  - Instructor
  - System
- Enter and submit username and password
  - Student
  - System
- Enter e-mail address (optional)
  - Student
  - System
- Inform instructor about student account creation
  - Instructor
  - System
- Give permission to the system for account creation
  - Instructor
  - System
- Create student account
  - Instructor
  - System
11.2.4. Annotation use-case diagram
11.2.5. Collaboration use-case diagram
11.2.6. File Manipulation use-case diagram

- Request to see existing files
  - Display existing files
  - <<includes>> (if display file is requested)

- Request to load a work from existing files
  - Load a work from existing files
  - <<includes>> (if load file is requested)

- Do Work Manipulation on files

- Request to see pre-saved files
  - <<includes>> (if see pre-saved file is requested)
  - See pre-saved files

- Request to save the file to the file structure
  - <<includes>> (if save file is requested)
  - Save the work to file structure
11.2.7. Presentation Manipulation use-case diagram

- **Request to see recent documents kept with respect to session.**
  - <<includes>> (if recent documents are requested to be seen)
- **Display recent documents kept with respect to session**
- **Request to start slideshow**
  - <<includes>> (if slideshow is requested to be started)
  - **Start slideshow**
  - Do annotation manipulation on slides
- **Instructor**
  - <<includes>> (if slideshow is started)
  - Manipulate slides by using swipe gesture (i.e., forward, backward, go to nth slide)
  - Grounds editing on slides
  - Add/remove/edit slides
  - <<extends>> (if slideshow is edited)
  - <<extends>> (if slideshow is edited)
- **Exit slideshow**
  - <<includes>> (if slideshow is requested to be saved)
  - Save (as) annotated/edited slideshow
11.2.8. Video Manipulation use-case diagram
11.2.9. Webpage Manipulation
11.3. Activity diagrams

11.3.1. IWBC activity diagrams

11.3.1.1. Initialization and configuration

11.3.1.2. Content display and annotation
11.3.1.3. Sketching

11.3.1.4. Collaboration
11.3.2. CBC activity diagrams

11.3.2.1. CBC login
11.3.2.2. CBC collaboration