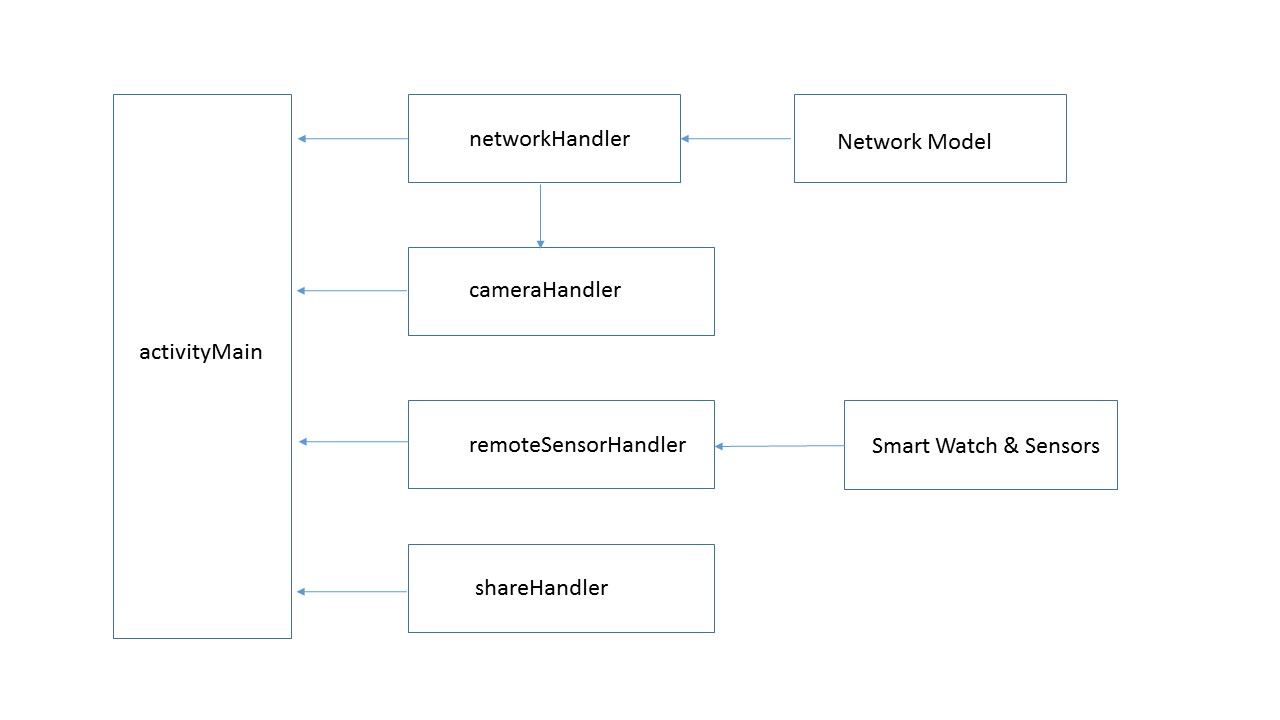
**CoMotion**

**Developer Guide**

Jiang Shenhao (A0119401)

1. System Requirement
2. Android SDK: 21 and above;
3. Android System 5.0 (Lollipop) or later;
4. System supports Wi-Fi Direct feature;
5. Android Wear (App): downloadable from Google Play Store;
6. Android Wear (System): 5.0 or above;
7. Android Wear (Smart watch): supports linear-accelerometer.
8. General Structure



1. Hardware Requirement and Manifest.xml

The application will make use of the following hardware components:

1. Wi-Fi module;
2. Back camera;
3. Internet module;
4. Storage component;
5. Android Wear (smart watch).

Therefore the following declaration needs to be made in the AndroidManifest.xml:

<uses-permission android:name="android.permission.INTERNET"/>

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/>

<uses-permission android:name="android.permission.ACCESS\_WIFI\_STATE"/>

<uses-permission android:name="android.permission.CHANGE\_WIFI\_MULTICAST\_STATE"/>

<uses-permission android:name="android.permission.CHANGE\_WIFI\_STATE" />

<uses-permission android:name="android.permission.CHANGE\_NETWORK\_STATE" />

<uses-feature android:name="com.android.hardware.camera2.full"/>

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.CAMERA"/>

<uses-permission android:name="android.permission.RECORD\_AUDIO"/>

<uses-feature android:name="android.hardware.type.watch" />

1. Main Packages (Handheld)

(Only important classes are provided here.)

|  |  |
| --- | --- |
| Package Name | Package Description |
| activityMain | Provides the list of commands the user may choose. |
| networkHandler | Establishes the Wi-Fi P2P network for communication. |
| cameraHandler | Enables the back camera for video recording. |
| remoteSensorHandler | Communicates with the smart watch for sensor reading. |
| shareHandler | Communicates with Google Drive for uploading process. |

1. activityMain Package:

|  |  |
| --- | --- |
| Class Name | Class Description |
| MainActivity.java | Listens to the user’s choice on the main UI interface. |
| ApplicationHelper.java | Defines some helping methods. |
| ActivityConstant.java | Constant values. |
| PauseResumeListener.java | Interface. |

1. MainActivity.java

The MainActivity.java inflates the main UI interface when the user launches the app. It also listens to the user’s choice of actions to take by setting onClickListener for various buttons on the main UI. The user may choose to start certain activity or service by pressing the corresponding button. For details refer to onCreate() method of the class.

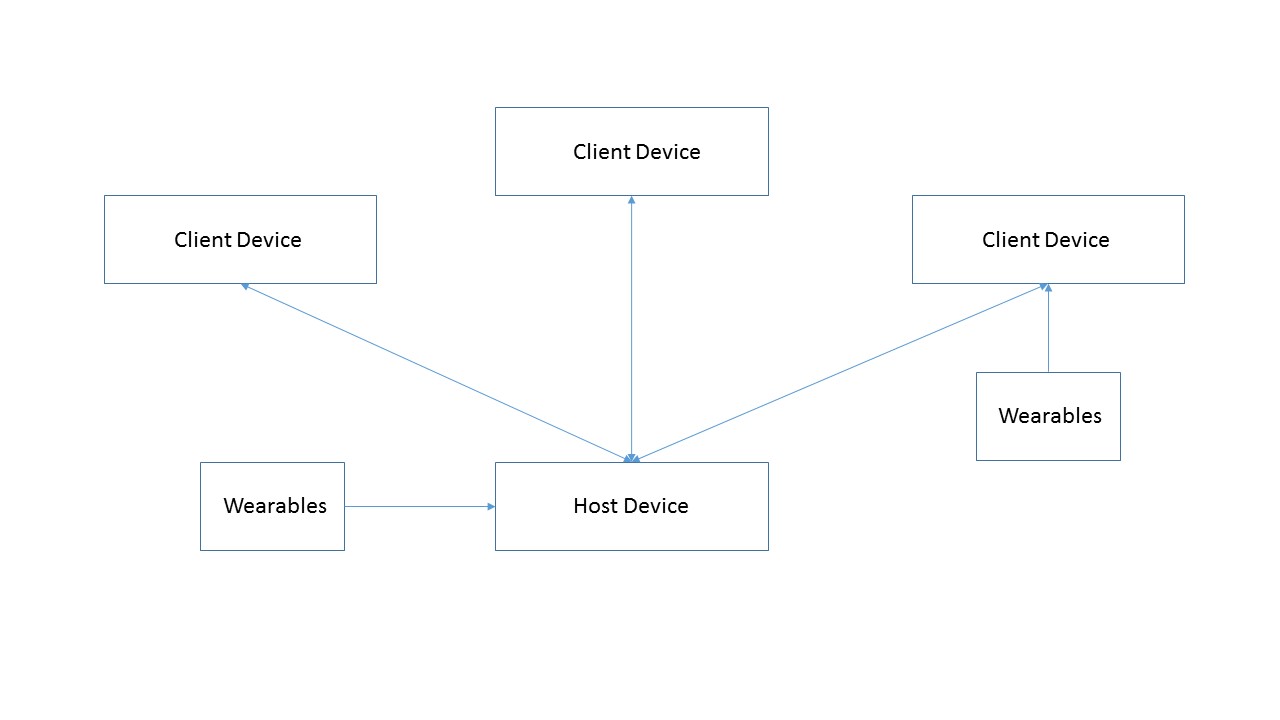
1. ApplicationHelper.java

The ApplicationHelper.java defines some useful utility methods which can be used by all parts of the program, including showing toast messages and returning the activity instance.

1. networkHandler Package:

|  |  |
| --- | --- |
| Important Class Name | Class Description |
| PeerSettingActivity.java | Checks for Wi-Fi P2P connection. |
| NetworkActivityTemplate.java | Activity class for P2P connections (important). |
| NetworkService.java | Main handler for sending and listening of messages. |

Network Model (Main Structure of the Wi-Fi Direct P2P Network) \*:



\*Note: The communication among handheld devices (host or client devices) is via Wi-Fi Direct P2P. In the graph, the communication to wearables is via Bluetooth. The reason for choosing Wi-Fi Direct instead of Bluetooth is that Wi-Fi Direct is able to cover more distance than Bluetooth, and the throughput of Wi-Fi Direct is also much larger than Bluetooth.

(1) PeerSettingActivity.java

The PeerSettingActivity.java inflates the main control view of peer connections. A list of available devices will be shown by this activity, and the user can choose to connect to, or disconnect from the selected device.

(2) NetworkActivityTemplate.java (Important!)

The NetworkActivityTemplate.java is the activity class used for Wi-Fi P2P connection. Basically any class which uses P2P connection needs to extend this class, instead of the normal activity. And inside the class, performConnectionDiscovery() method needs to be overridden as:

protected boolean void performConnectionDiscovery() {

return true;

}

1. NetworkService.java

The NetworkService.java defines two of the most important functionalities of P2P communication: sending and receiving of the messages.

Important notes:

public static interface MessageHandleListener {};

public abstract void sendMessage(NetworkMessageObject);

These are the main interfaces for sending and receiving messages. For classes using P2P communication, MessagehandleListener needs to be overridden to finish a certain task defined by the developer (for example, start recording videos in CameraFragment.java). for sending messages, there are two options: a message can be sent either from point to point (one device to one device), or it can be broadcasted in the network. For broadcast of messages, the target IP address is 255.255.255.255. By sending to this address, the message will be broadcasted inside the P2P network.

1. cameraHandler Package:

|  |  |
| --- | --- |
| Class name | Class Description |
| CameraActivity.java | Main activity holding the fragment component. |
| CameraFragment.java | Fragment defining most of the functionalities. |
| AutoFitTextureView.java | Helper class. |
| MusicFlagService.java | Generates a sound to indicate that recording has started. |

1. CameraActivity.java

The activity holds the fragment (CameraFragment.java) and overrides the performConnectionDiscovery method.

Important note: the cameraHandler uses the network template provided by the networkHandler, and as the template requires, any activity which may use the Wi-Fi P2P connection needs to override the performConnectionDiscovery method from NetworkActivityTemplate.java.

1. CameraFragment.java

This fragment is the main place taking charge of the camera behaviors. The original code can be found from the Camera2Video sample provided by Android Studio. It is modified in the following aspects:

* The storage location and naming format are changed: a folder named as ‘CoMotion’ will be created inside the PICTURE folder in the system, and each video file is named based on the time stamp (absolute time, not System.currentTimeMillis()). For details refer to getVideoFile() of the class.
* Communication via Wi-Fi P2P network is enabled: the user can choose to record videos on one single device, or on all devices connected to the network. For details refer to onClick(), NetworkService.MessageHandleListener, onCreate() of the class.
* Communication to the Android Wear smart watch is enabled: when the user chooses to record videos on all connected devices, a signal will be sent to start the smart watch (if it is available) at the same time, and start recording data. For details refer to onClick() of the class.
* When multiple devices are connected, after the starting signal is given out, the host device will generate a beep sound, so even if the devices fail to start recording videos simultaneously, later the sound will help the user to align videos in the video editing software. For details refer to onClick() of the class.

Important notes:

private void startRecordingVideo();

private void stopRecordingVideo();

Note that these two methods interact with the UI of the activity directly: the texts appearing on the buttons will be changed by these two methods. Therefore, actions related to it must be placed inside: runOnUiThread(new Runnable() {}). This tip might be useful for future development.

(3) MusicFlagService.java

A service running in the background of the host device. Three seconds after the start recording signal is given out, the host device will generate a beep sound. Later when the user wants to align the videos with the video editing software, he can simply align them according to the sound wave.

1. remoteSensorHandler Package:

|  |  |
| --- | --- |
| Class Name | Class Description |
| WearabelMessageService.java | Sends commands to the wearable device. |
| DataStorageService.java | Stores received data from the wearable. |
| Utils.java | Constant values. |
| WearPatternActivity.java (Incomplete) | Activity used for recording patterns. |

(1) WearableMessageService.java

The WearableMessageService.java is a service class used to send command to the wearable based on the choice of the user. The command types can be start, stop or record pattern. GoogleApiClient and Wearable.MessageApi are used, so it may require further reading if the developer is not familiar with Android Wear.

(2) DataStorageService.java

The DataStorageService.java is a listener service used to listen for the data transmitted from the wearable to the handheld device. Note that the data transmitted back is in DataMap format, and it’s casted to JSONObject in this service. The produced JSONObject is formatted to “x-value,y-value,z-value,time\_stamp” pattern and then stored in the handheld as a text file. Similar to the storage of video files, a folder named as ‘CoMotion’ will be created inside DOCUMENT section of the system. Each text file is named based on time stamp (System.currentTimeMillis(), not absolute time. This is to be coherent with the data it contains: the time\_stamp of the transmitted data is in System.currentTimeMillis() format).

(3) WearPatternActivity.java (Incomplete)

Note: here the coding is finished, but it hasn’t been tested yet due to time constraint, therefore it is marked as incomplete.

The WearPatternActivity.java is the activity used to record a certain pattern of the wearable. Here the pattern means a set of movement of the user. Basically the user can choose the delay time (how long to be prepared) and the recording time (how long the movement will be).

1. shareHandler Package

|  |  |
| --- | --- |
| Class Name | Class Description |
| UploadActivity.java | Uploads the files to Google Drive. |

(1) UploadActivity.java

The UploadActivity.java is the activity used to upload the files (including videos and text files) to the user’s Google Drive account. Once the device is connected to the Google Drive account, the user will be able to choose the file to upload. Again the GoogleApiClient is used.

1. Main Package (Wearable)

|  |  |
| --- | --- |
| Class Name | Class Description |
| WearDataCollector.java | Main service collecting sensor data on the wearable. |
| Utils.java | Constant values. |

(1) WearDataCollector.java

The WeraDataCOllector.java is the main service running on a wearable to collect the sensor data. It extends the WearableListenerService for listening for the event changes, and implements the SensorEventListener for sensor data reading. The data is read from the sensors as DataMap format, and transmitted to the handheld via GoogleApiClient interface.

Important notes:

onSensorChanged(final SensorEvent event);

This is the main method for the program to interact with the sensors on the wearables. The sensors are very sensitive, so any changes on the readings of the sensors will trigger the sensor to send the newest data to the handheld, and here by setting TRANSMISSION\_GAP, the developer can actually control the frequency of sending the data. Note that empirically the maximum frequency is 24 Hz, meaning that in one second the maximum number of data which can be transmitted from the sensors to the handheld device is 24, which is more than enough to analyze the pattern of the movement.

1. Future Improvements

a. Better UI

The main UI design principle of the app is material design, as proposed by Google for Android 5.0 Lollipop. By setting the style.xml and color.xml, CoMotion roughly follow the pattern. However, due to time constraint, the layout of the app may not be good enough for devices with small screens (the app is tested mostly on a Nexus 9 tablet, so the UI looks okay on it; but it may have some problems on smartphones).

b. Finish WearPatternActivity

As stated in the remoteSensorHandler package, this activity is not tested yet, although the coding part is finished. The main function of the activity is to record a certain pattern of the movement, so essentially this activity is just delaying the wearable for some time, and then start and stop reading automatically.

c. More Robust networkHandler

This Wi-Fi P2P network handler is migrated from a previous project, and by testing it seems that it’s not robust enough. Sometimes the command from the host device cannot be received by every client, and the delay is quite long. After the host device gives out the command, sometimes it takes up to several seconds before the client device receives it. This will be elaborated more in the next section: Known Issues.

1. Known Issues

a. networkHandler Package

The developer of the handler stated that the host device needs to start the app first, then the client devices; otherwise the app will crash. Another problem is the delay: if the host device sends out the command, the client devices may not be able to receive it instantaneously; if one of the client device sends out the command, the delay might be even worse.

b. shareHandler Package

If the file size is too big (several hundred MB), the app may crash due to memory issues. It has been tried to read the data into the buffer byte by byte, but the problem still occurs. As the result, it is recommended that video files can be exported directly from the device, instead of uploading it to Google Drive.

c. Sensors and Wearables

By experiment, the watch is able to collect data for a maximum of 15 minutes. This may be due to the cache problem that the smart watch needs to clean up its system if the amount of data is too big.

1. Further Readings

There are a handful of Github repositories which might be helpful for understanding how this project works, as well as for further development:

Sensor Dashboard:

This project is very useful for understanding how sensors on Android Wear works.

<https://github.com/pocmo/SensorDashboard>

Android Wear Data Collection:

This project can be used as a complement for Sensor Dashboard: it shows in detail how to transmit the data from the wearable to the handheld device.

<https://github.com/gjacobrobertson/android-wear-data-collection>

Android Google API Quickstart:

This project is the training project showing developers how to use the GoogleApiClient to connect to Google Drive and upload files.

<https://github.com/googledrive/android-quickstart>