임베디드시스템소프트웨어 프로젝트 보고서

20171643 박준혁

DATE: 06/26 2023

프로젝트 이름:	**ootdPT – Weather Information based Outfit Recommendation Android Application on archo-i.mx6q device**
프로젝트 요약:	IoT(Internet on Things) 기술이 가정집으로 들어오며 등장한 '스마트 인터폰'에, 최근 각광받고 있는 ChatGPT 인공지능 기술을 접목시켜 **'날씨에 따른 옷차림 추천 서비스'**를 제공하고자 한다.

Motivation

- **사물인터넷(IoT, Internet on Things)** 기술은 등장과 함께 많은 분야에서 활용되며 인간의 삶의 질을 향상시키게 되었다. 그 중 가장 대표적인 적용 분야는 이른바 '스마트 홈'이라며 불리는 '네트워크 기반의 가정집 단위 스마트 컴퓨팅 환경'이었다. 집안 내부 공기의 질을 분석해 공기 청정기가 자동으로 작동된다던가, 또는 스마트폰만으로 집안 내부의 다양한 전자기기를 통제하는 등, 스마트 홈 개념은 현재 많은 가정 내 전자기기에 도입되어 활용되고 있다.

- 이때**, '스마트 인터폰'**이라 불리며, 주로 가정집 거실에 설치되어 집안 내부의 에어컨, 세탁기, 전화기, TV 등을 통제하고, 나아가 (아파트의 경우) 집안 외부의 엘리베이터가 현재 어느 층에 위치해있는지 등도 확인할 수 있는 IoT 임베디드 시스템이 등장하였다**. 2010년대 초부터 지어진 대부분의 국내 가정집은 이러한 '스마트 인터폰'을 탑재**하고 있다.

Ex) Intercom

COMMAX SmartHome & Security

KOCOM

- 한편, 2022년 말 공개된 **OpenAI의 ChatGPT**는 기존 대화형 인공지능 서비스의 성능을 월등히 뛰어넘는 성능을 보이며 단순히 업계뿐만 아니라 국제 사회 전체의 주목을 받고 있다. 현재 ChatGPT는 출시가 1년이 채 되지 않았음에도 이미 다양한 분야에서 널리 활용되고 있으며, 동시에 그러한 ChatGPT를 기반해 2차적인 인공지능 서비스를 제공하는 Application도 서서히 등장하고 있다.
- 이때, 본인은 고성능의 **ChatGPT**가 앞서 언급한 **'스마트 인터폰'과 결합**하면 보다 더 현대인의 삶을 개선시킬 수 있다는 생각 하에 'ootdPT' 프로젝트를 제안한다. **ootdPT는 스마트 인터폰 내에서 돌아가는 Subprogram으로서, 네트워크 상에서 실시간 날씨 정보를 얻어온 후 해당 정보를 토대로 ChatGPT에게 '그날에 맞는 옷차림'을 질문하고, ChatGPT가 추천한 옷차림을 사용자에게 알려주는 서비스**이다. 참고로 'ootd'라 함은 'Outfit Of The Day'의 줄임말로, 우리말로 '오늘의 옷차림'이라는 뜻을 가리킨다.

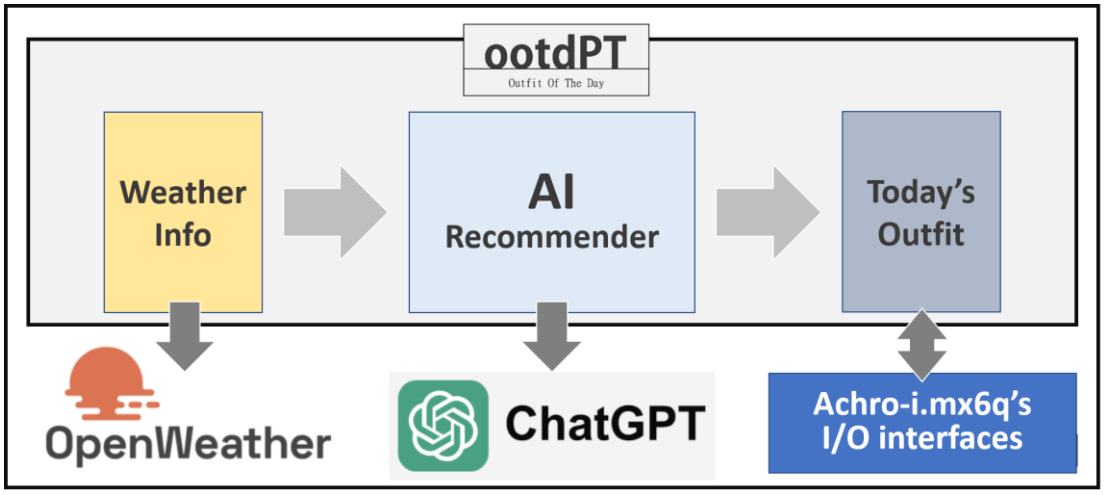
Project - ootdPT

ootdPT

Outfit Of The Day

Figure - ootdPT's logo

- **‘ootdPT’는 Java로 작성된 Android Application**으로, 우리 수업의 실습용 보드인 ‘achro-i.mx6q’ Model Device에서 돌아가는 **‘날씨에 따른 옷차림 추천 서비스’**이다. ootdPT는 다음과 같은 구조로 표현할 수 있다.



- Application은 시작과 함께 네트워크로부터 **Weather Information을 추출**한다.
  - 날씨 정보는 대표적인 무료 날씨 제공 API인 openweathermap의 API를 사용하여 데이터를 가져온다.
- 이어, 추출한 Weather 정보를 Outfit Recommendation 단계로 넘긴다. 해당 단계에서는 **ChatGPT API를 이용해 날씨에 알맞은 옷차림 추천을 생성**한다.
- 생성된 ‘옷차림 추천’ 및 ‘날씨 정보’ 등은 Device의 여러 I/O Device(FND, LED, TEXT_LCD, DOT)에 출력되어 User가 시각적으로 바로 알아볼 수 있게 된다.

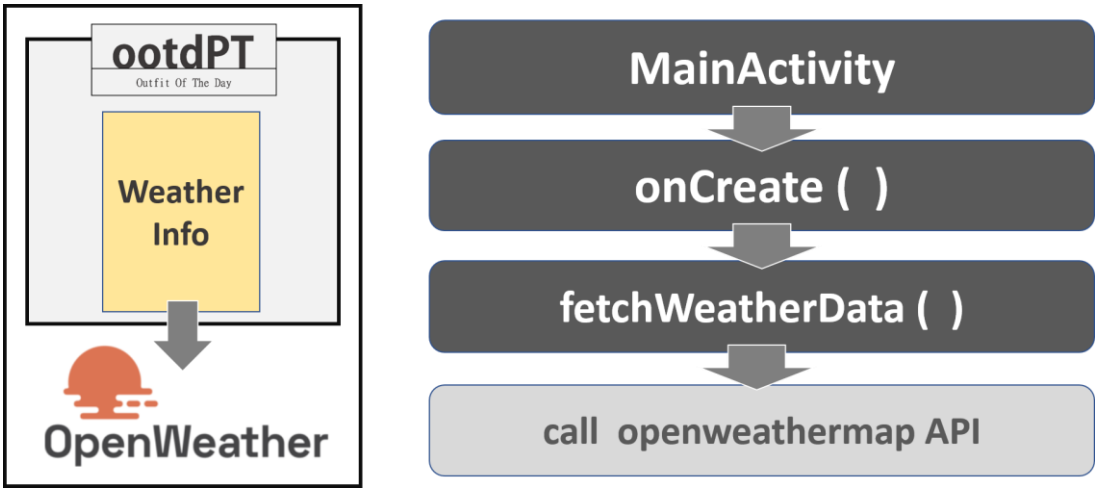
Development

- 지금부터 ootdPT의 개발 과정을 소개한다. 개발 과정은 상기 ‘구조’ 그림을 세 단계로 나누어 단계별로 설명한다. 단계 구성은 다음과 같다.
  - **1단계: Weather Information 추출**
  - **2단계: Outfit Recommendation 생성**
  - **3단계: Device Communication 과정**
- 우선, 1단계인 Weather Information 과정부터 확인하자. 참고로 ootdPT는 일반적인 Android Application 개발 과정을 준수하고 있으며, 후술할 용어는 그러한 관점

에서 해석할 수 있다. (ex. MainActivity, Layout, AndroidManifest 등)

1. Weather Information 추출

- 이 과정은 다음과 같은 구조로 설명할 수 있다. 우선 그림을 보자.



- ootdPT의 시작 Entry Point 및 Main Program 역할을 하는 MainActivity.java에선 onCreate( )의 동작과 함께 일부 초기화 작업 직후 바로 **'fetchWeatherData'**라는 이름의 함수를 호출해 Weather Information 추출 작업을 시작한다.

- 이어, fetchWeatherData 함수에선 Network Connection을 위한 초기 세팅 작업을 수행 후, 이어 openweathermap API를 호출해 날씨 데이터를 받아온다.

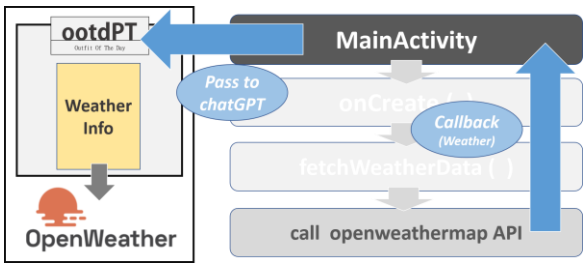
  ■ 이때, 날씨 데이터의 지역 기준은 **서울**이다.

  ■ 날씨 데이터에는 기본적인 **기온뿐만 아니라 습도(Humidity)와 '날씨 요약(ex. Clear Sky)'이 포함**된다.

  ■ 이 과정에서 현재 **'날짜' 데이터도 마련**한다. 이는 Java Language에서 기본으로 제공하는 함수를 활용한다.

  ◆ 즉, Device에 설정된 시간 및 날짜를 기준으로 한다.

  ■ 이러한 날씨 데이터 추출은 추출 시점을 특정할 수 없기 때문에 비동기적으로 동작하며, 날씨 데이터 추출 직후 **MainActivity는 Callback을 받아 이어지는 작업을 수행**한다. 이 과정은 우측 그림으로 묘사할 수 있다.



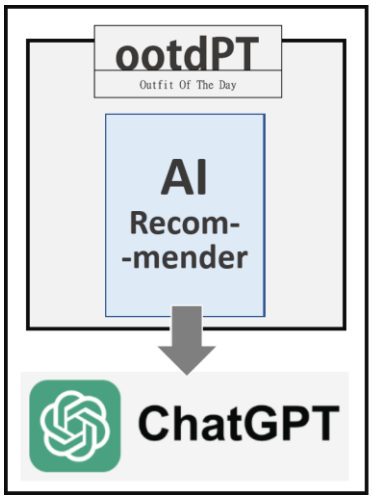
```java
private void fetchWeatherData() {
    String openWeatherMapApiKey = "                                ";
    String city = "Seoul";

    String apiUrl = "http://api.openweathermap.org/data/2.5/weather?q=" + city + "&appid=" + openWeatherMapApiKey;

    WeatherTask weatherTask = new WeatherTask();
    weatherTask.execute(apiUrl);
}

private class WeatherTask extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... urls) {
        try {
            URL url = new URL(urls[0]);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.connect();

            InputStream inputStream = connection.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            StringBuilder result = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                result.append(line);
            }

            return result.toString();

        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);
        if (result != null) {
            try {
                JSONObject json = new JSONObject(result);
                JSONObject main = json.getJSONObject("main");
                double temperature = main.getDouble("temp");
                int humidity = main.getInt("humidity");
                String weatherDescription = json.getJSONArray("weather").getJSONObject(0).getString("description");

                double temperatureCelsius = temperature - 273.15; // Convert temperature from Kelvin to Celsius
                int roundedTemperature = (int) Math.round(temperatureCelsius); // Round the temperature to the nearest integer

                Args temp = new Args();
                temp.date = getCurrentDate();
                temp.humid = humidity;
                temp.tempCelsius = temperatureCelsius;
                temp.weatherDescription = weatherDescription;
                fd = openDevice(temp); // Open the driver with humidity, temperature, and weather information

                // Call the method to get outfit recommendations
                recommendOutfit(weatherDescription, roundedTemperature, humidity);

            } catch (JSONException e) {
                e.printStackTrace();
                showErrorToast("JSON parsing error");
            }
        } else {
            showErrorToast("Failed to fetch weather data");
        }
    }
}
```

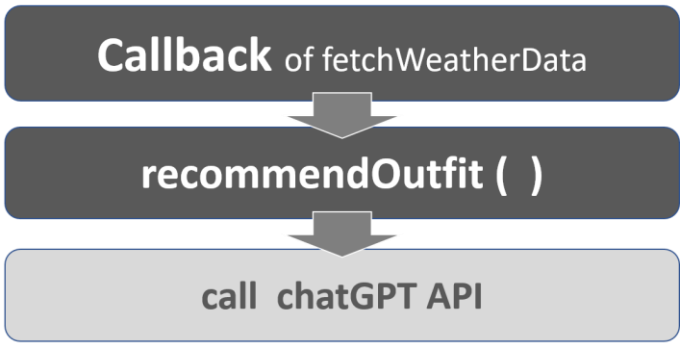
- **위는 fetchWeatherData 함수 코드, 그리고 기타 관련 루틴의 코드**이다.
- HttpURLConnection Class를 이용해 openweathermap API 사용 주소로 이동하고, 거기서 JSON 형식의 Data를 읽어 들여 Parsing하고 있음을 알 수 있다.
  - 참고로, 위 API 호출이 제대로 동작하기 위해선 Device에 **WIFI 수신기 혹은 Ethernet선을 연결**해 네트워크 연결을 꼭 보장해야 한다.

- 이때, Weather Data Parsing 이후에는 다음과 같은 작업이 이뤄지고 있음을 주목하자. 이들은 모두 후술할 내용과 직접적으로 연결되는 부분이다.
  - ■ **'dev_driver'**라는 본 ootdPT의 Device I/O를 위해 반드시 필요한 Driver를 함수 openDevice를 통해 open한다.
    - ◆ 당연히 이는 JNI(Java Native Interface)를 사용해 구현해야 하며, 자세한 내용은 세 번째 단계 설명 때 이어간다.
  - ■ 'recommendOutfit'이라는 이름의 함수를 호출해 **Outfit Recommendation**을 수행한다.
- 이렇게 해서 첫 번째 단계 작업이 마무리된다. 이어 다음 단계를 보자.

2. Outfit Recommendation 생성

- 바로 위에서 확인한 바와 같이, Weather Information을 추출한 직후 MainActivity는 'recommendOutfit' 함수를 호출해 '날씨에 따른 옷차림 추천'을 수행한다. 이 단계는 **아래와 같이 수행**되어야 한다(그리고, 그렇게 될 계획이었다).





- 허나, 이 부분에서 **큰 문제가 발생**했다. 바로 현재 우리 실습 보드 achro-i.mx6q Device에 탑재된 Android Version 4 Jelly Bean 및 Host PC의 Eclipse(Java 6)가 ChatGPT API를 지원할 수 없는 것이었다.
- 그 배경엔 몇 가지 이유가 있었는데, 간단히 요약하면 다음과 같다.
  - ■ ***현재 OpenAI에선 Java용 API를 '공식적으로' 지원하고 있지 않다.***
  - ■ *민간에서 개인 프로젝트 형식으로 Java용 Customized OpenAI API Package를*

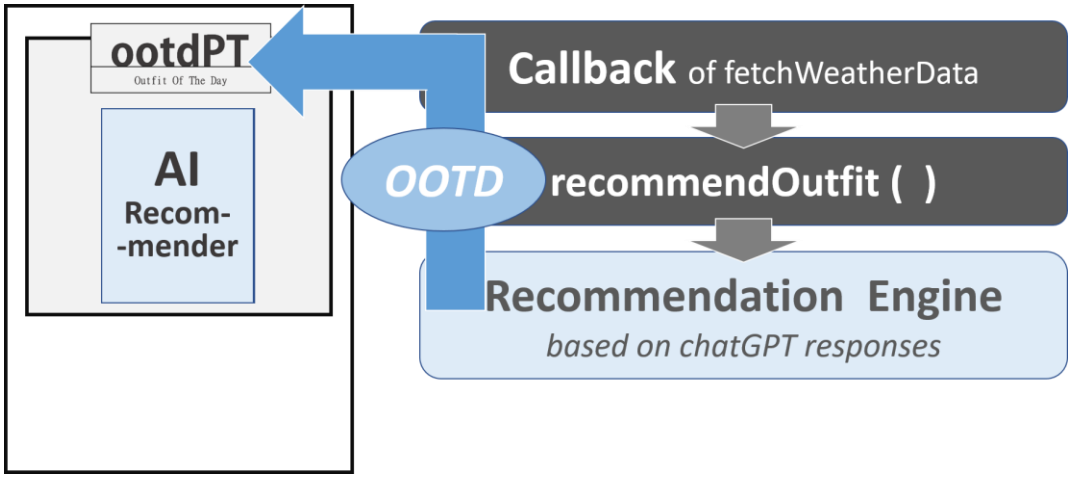
출시한 바 있지만, 이들은 모두 ***현재 우리 개발 환경보다 최소 하나 또는 둘 이상의 상위, 최근 버전의 Java와 Android를 요구****한다(Java: 8 이상, Android: Lollipop 이상).*

- ◆ *현재 OpenAI API에서 사용하는 SSL/TLS Certificate Validation 방식은 구형 Java 및 Android에서 지원하지 않고 있다.*

- 따라서, 결국 차선책이 필요했고, 본인은 다음과 같은 방안들을 떠올려보았다.
  - **방안1)** In-Device AI-like Recommendation Engine 구현
  - **방안2)** Intermediate Server Approach (Network기반 또는 Serial 연결 기반)
  - **방안3)** Device 내 Android Version Upgrade(또는 수정)

- 이 중 **방안1과 방안2는 당장의 구현이 가능**하지만 방안3의 경우 여러 의존성 문제와 'Device 자체 내부 System을 훼손할 수 있음'을 고려해 본 프로젝트에선 제외하기로 결정했다.
  - **한 가지 확실히 해둘 것은, 본인은 방안3이 '불가능'한 것은 아니라고 생각한다. 다만 현재 본인의 네트워크 및 Android 지식 수준을 고려해 해결책에서 제외한 것일 뿐**이다.

- 따라서 본인은 방안1과 방안2를 구현하기로 하였으며, 이들을 따로 별도의 프로젝트로 구분하지 않고, 둘을 하나의 프로젝트에 동시에 구현하였다.
  - 즉, **현재 제출된 ootdPT는 방안1과 방안2를 모두 동시에 수행**하고 있다.

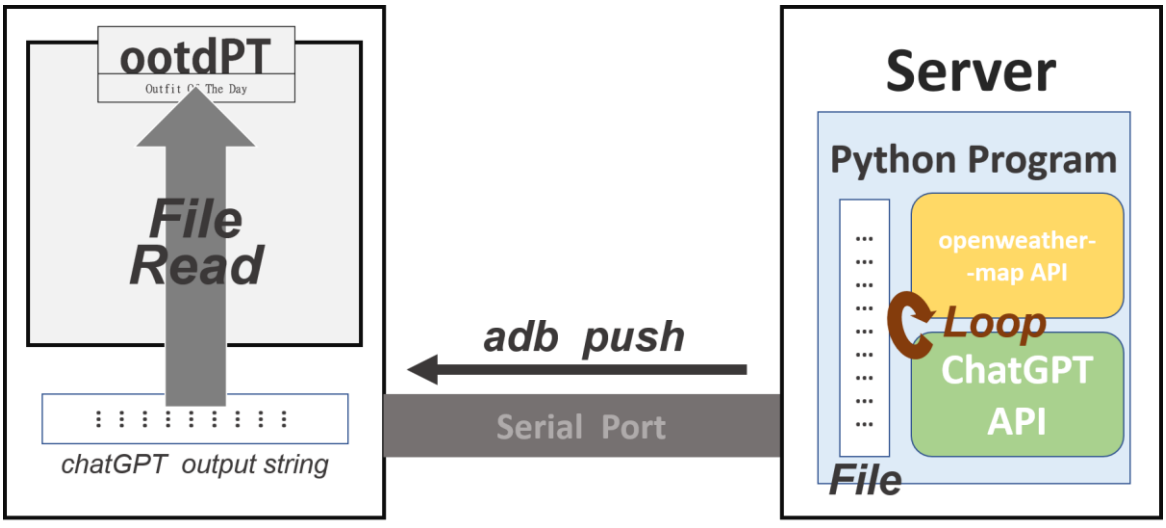


- 먼저 **방안1의 구현**에 대해 알아보자. 위의 그림이 이 과정을 묘사하고 있다.
  - recommendedOutfit 함수 내부에서 자체적으로 **날씨 데이터를 분석해 일련의 기준을 두어 상황(Case)에 맞는 Outfit을 Deterministic하게 출력**한다. 아래의 코드와 같이 말이다(코드 일부만 포함).

```
private void recommendOutfit(String weatherDescription, int temperature, int humidity) {
    // Perform outfit recommendation based on weather description, temperature, and humidity
    String outfit;

    if (weatherDescription.contains("rain")) {
        if (temperature < 10) {
            outfit = "Raincoat, boots, and a warm jacket";
        } else if (temperature < 15) {
            outfit = "Raincoat and boots";
        } else {
            outfit = "Rain jacket and waterproof shoes";
        }
    } else if (weatherDescription.contains("cloud") || weatherDescription.contains("overcast") ||
```

  - 이렇게 결정된 Outfit Recommendation을 새로운 TextView로 화면에 출력한다.
  - 굉장히 간단한 Engine이지만, **해당 (Fixed) Outfit들은 모두 ChatGPT를 여러 차례 실험하여 결정한 내용이기 때문에 'AI-like'하다고 할 수 있다.** 본 프로젝트는 상업용 Application 개발이 목적이 아니기에 본인은 약 20개의 Case만을 구분하였지만, 만약 ChatGPT를 사용하지 않은 채 이러한 Service를 실제 상업 수준으로 제공한다면 더 많은 Case를 두어 구현할 수 있을 것이다.

- 이어 **방안2의 구현**을 보자. 방안2는 아래와 같은 구조로 동작한다.



  - Server에서는 openweathermap API와 ChatGPT API를 활용해 현재 ootdPT의 **주요 Service를 모두 수행하는 Python 기반 프로그램이 동작**한다(OpenAI에

서 Python용 API Package를 공식적으로 제공하고 있다).

◆ 이 프로그램은 Server(Cross-Compile 환경 기준 Host PC)에서 동작하며, 'Weather Information 추출 -> ChatGPT Outfit Recommendation 생성'의 과정을 5초마다 주기적으로 반복한다.

◆ 그리고 매 반복마다 **'생성된 Outfit Recommendation'**을 Text File의 형태로 **Target Device에 ADB PUSH**한다. File명은 'ootdPT_gpt.txt'로 고정한다.

■ Device의 ootdPT Application(MainActivity)은 onCreate에서 다음과 같은 액션을 정의한다. App이 실행된 상태에서 User가 특정 Button을 클릭하면 Device 내부의 (Host PC에서 받은) 'ootdPT_gpt.txt'를 읽어 ChatGPT가 생성한 Outfit Recommendation을 받는 것이다.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    linearLayout = (LinearLayout) findViewById(R.id.container);

    // Bind to the service
    Intent serviceIntent = new Intent(this, MyService.class);
    bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);

    // Call the method to fetch weather data
    fetchWeatherData();

    // Set up the button click listener
    Button openFileButton = (Button) findViewById(R.id.openFileButton);
    openFileButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            openFile();
        }
    });
}
```

■ openFile 함수에선 'ootdPT_gpt.txt' 파일을 읽은 후 해당 파일 내용물을 화면에 TextView 형태로 뿌려주는 작업이 이뤄진다.

- 이러한 방안2는 실제 상업용 임베디드 시스템에서 ChatGPT를 사용하고자 하지만 직접적으로 API를 호출하기엔 내부 탑재 Android의 Version Issue가 있는 상황에서 적용할 수 있는 방안이다. 위에서 확인한 것과 같은 동작을 수행하는 **Server를 두어 Serial Port 또는 Network를 통해 Remote Server에서 이뤄진 Service를 넘겨 받아 User에게 Service를 제공**하는 것이다.
- 결론적으로 이렇게 두 방안을 동시에 구현해 이들이 모두 실제로 작동하고 있다.

3. Device Communication 과정

- ootdPT는 본 실습 보드의 **I/O Device 중 FND, LED, TEXT_LCD, DOT를 활용**한다. 물론 실제 상업용 임베디드 시스템 중 본 실습 보드에 탑재된 수준의 FND, LED, LCD, DOT Device를 사용하는 경우는 흔치 않지만, ootdPT는 이들을 실제 수준의 Device라 가정하였다.

- 이를 위해 실습 보드에는 **'dev_driver'**라는 디바이스 드라이버가 설치되어 있다. 이 Driver는 **FND, LED, TEXT_LCD, DOT를 동시에 제어**하며 ootdPT가 원하는 출력을 수행한다. 각 Device에서의 출력물은 다음과 같다.

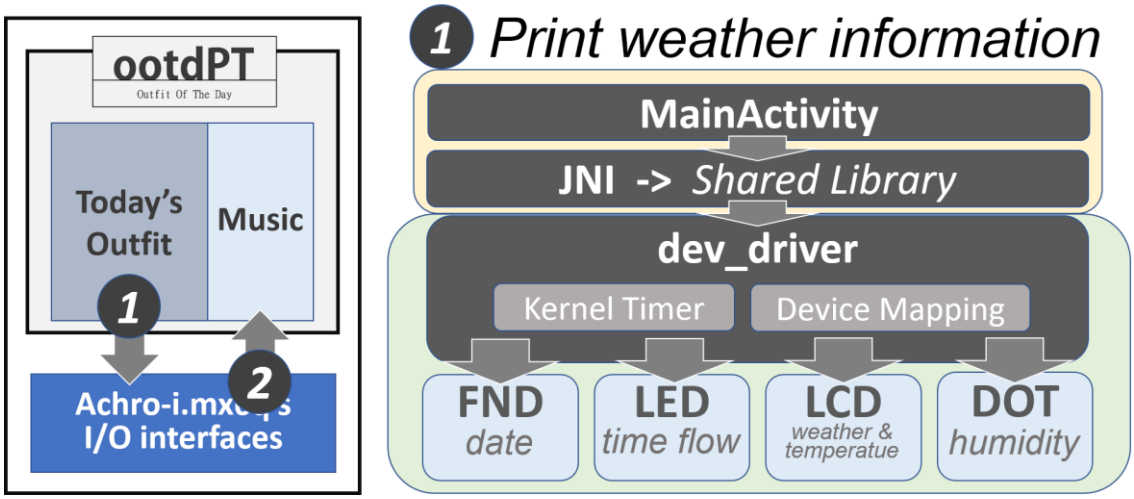
  - **FND:** 날짜(Date) 정보 ex) 1월 2일 -> 0102
  - **LED:** 시간 흐름 표현 ex) 매초마다 특정 패턴을 따라 깜빡임
    - *이를 위해 'dev_driver'에는 **Kernel Timer**가 설치되어 있고, 1초를 주기로 연쇄적으로 Timer 등록 및 Handler 수행이 이뤄진다.*
  - **TEXT_LCD:** Weather Description 및 기온(Temperature)을 출력
  - **DOT:** 습도(Humidity)를 출력 ex) 74% 습도 -> 7
    - openweathermap API에선 습도가 % 단위로 제공되는데, 그 십의 자리 수를 DOT Device에 출력하는 것이다. User는 DOT에 출력된 숫자를 보고 그날의 습도를 알 수 있다.

```
JNIEXPORT jint JNICALL Java_com_example_androidex_MainActivity_openDevice(JNIEnv *env, jclass this, jobject
    int fd;
    struct _Args {
        char date[MAX_LEN];
        int humid;
        double tempCelsius; // Updated data type to double
        char weatherDescription[50]; // Added weather description
    } arg;
    const char *date;
    char ioctl_param[MAX_LEN+8+50]; // Increased ioctl_param size to accommodate weather description

    /* Open the device driver */
    if ((fd = open(dev_file_addr, O_WRONLY)) == -1) {
        __android_log_print(ANDROID_LOG_VERBOSE, "DEV_DRIVER", "Failed to open device driver");
        return -1;
    }

    /* Data assignments */
    jclass argsClass = (*env)->GetObjectClass(env, args);
    jfieldID jdateID = (*env)->GetFieldID(env, argsClass, "date", "Ljava/lang/String;");
    jfieldID jhumidID = (*env)->GetFieldID(env, argsClass, "humid", "I");
    jfieldID jtempID = (*env)->GetFieldID(env, argsClass, "tempCelsius", "D"); // Updated data type to D (do
    jfieldID jweatherID = (*env)->GetFieldID(env, argsClass, "weatherDescription", "Ljava/lang/String;"); //

    // Date ex) 06/19
    date = (*env)->GetStringUTFChars(env, (*env)->GetObjectField(env, args, jdateID), 0);
    strncpy(arg.date, date, MAX_LEN);
    (*env)->ReleaseStringUTFChars(env, (*env)->GetObjectField(env, args, jdateID), date);

    // Humid ex) 64%
    arg.humid = (*env)->GetIntField(env, args, jhumidID);

    // Temperature
    arg.tempCelsius = (*env)->GetDoubleField(env, args, jtempID); // Updated data type to GetDoubleField

    // Weather description
    const char *weatherDesc = (*env)->GetStringUTFChars(env, (*env)->GetObjectField(env, args, jweatherID),
    strncpy(arg.weatherDescription, weatherDesc, 50);
    (*env)->ReleaseStringUTFChars(env, (*env)->GetObjectField(env, args, jweatherID), weatherDesc);

    /* Set parameters and then pass them */
    sprintf(ioctl_param, "%03d%03d%s%s", (int)arg.tempCelsius, arg.humid, arg.date, arg.weatherDescription);
    ioctl(fd, IOCTL_SET_OPTION, ioctl_param);  // Pass option parameters to device
    ioctl(fd, IOCTL_COMMAND);                  // Start the timer device!

    return fd;
}
```

- 'dev_driver'를 돌리기 위해 본 ootdPT Application은 **JNI를 활용**한다. dev_driver를 직접적으로 열기 위한 함수가 Java 언어 Level에선 제공되지 않기 때문이다. 따라서 이를 위한 open, close 등의 함수를 좌측과 같이 정의한다.

- 강의자료에서 배운 방식과 동일한 방식으로 Caller(MainActivity)로부터 **Data(날짜, 날씨 요약, 온도 등)를 받아오고 있음**을 확인할 수 있다.

- ‘ndk-build’를 통해 이러한 JNI 정의를 **Shared Library화**한 후, 이것을 ootdPT의 MainActivity에서 Load하여 ootdPT가 ‘dev_driver’를 접근 및 제어할 수 있다.

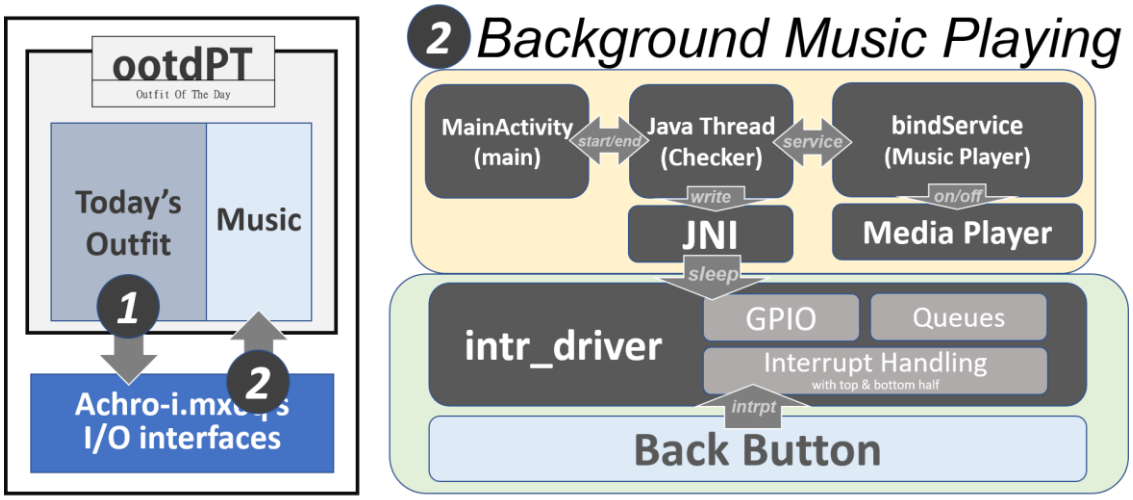


- 위 그림은 지금까지 설명한 내용을 직관적으로 보여주고 있다.

- 한편, ootdPT는 지금까지 설명하진 않았지만, 배경 음악도 실행할 수 있다. 본 서비스가 ‘스마트 홈’을 고려했음을 앞서 밝혔는데, 대부분의 이러한 서비스는 오전에 기상 직후 출근 전에 동작하면 가장 좋다고 할 수 있다.

  - 이때, ootdPT의 활용 시간대가 대부분 **‘기상 직후, 출근 전’일 것임을 고려해 Application이 차분한 음악을 배경으로 깔면 서비스 차원에서 좋을 것이라 판단**했고, 이에 따라 배경음악 기능도 추가하였다.

  - 이는 Java의 bindService와 Thread를 활용해 구현하였다. 우선, 아래의 그림을 보자.



- MainActivity는 Java Thread와 Java bindService를 동시에 띄운다. 그 둘의 역할은 다음과 같다.
  - **Java Thread: Device의 'Back Button' 입력에 대한 Listener 역할**을 한다.
  - **bindService:** 음악을 실행하거나 멈추며, Thread에서 인식한 'Back Button' 입력에 따라 ON/OFF가 제어된다. 그 와중에 MainActivity에서도 Flow의 시작과 끝에서 Service를 제어할 수 있다.
- 이때, Java Thread의 'Listening'은 어떻게 구현할까. 여러 방식이 존재하지만 본인이 선택한 방식은 **'Thread가 매번 Driver에 Write을 시도하고, Driver에선 Write을 시도한 Thread를 Sleep시킨 후 Interrupt(Back Button 입력) 발생 시에 이들을 깨워주는 방식'**이다. 이것이 CPU Core 사용 측면에서 (경미하지만) 더 경제적이라 판단했다.
  - 따라서, Java Thread에선 Loop를 돌리며 매번 이 Driver(intr_driver라고 함)에 write함수를 호출해 Write을 시도한다.
  - 매 Write 시 Driver에선 Write을 시도한 Thread를 (Interruptible) Sleep 시킨다.
  - 한편, intr_driver에선 동시에 'Back Button' 입력에 대한 GPIO를 기반으로 Interrupt를 설치해 'Back Button 입력'을 추적/관리한다.
    - 이때, Back Button Interrupt Handler는 Java Thread를 깨운다.
  - 그렇게 깨어난 Java Thread는 곧바로 bindService와의 Binder IPC를 통해 음악이 켜져 있으면 끄고, 반대로 음악이 꺼져 있으면 키는 기능을 수행한다.

- 이러한 일련의 작업이 유기적으로 동작해 ootdPT의 배경음악 서비스를 제공한다. 아래는 이 과정에 참여하는 코드 부분을 순차적으로 (주요 부분 위주로) 보여준다.

```
private ServiceConnection serviceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        MyService.MyBinder binder = (MyService.MyBinder) service;
        myService = binder.getService();
        isServiceBound = true;

        // Start the turn on/off thread after the service is bound
        turnOnOffThread = new TurnOnOffThread(myService);
        turnOnOffThread.start();
    }
```

◆ MainActivity는 bindService와 Java Thread를 동시에 띄운다.

```
@Override
public void run() {
    while (isRunning) {
            writeDevice(fd);  // It will sleep here!
        toggleMusic();    // After the thread woke up
    }
}
```

◆ Java Thread는 App이 종료되지 않는 한 매번 'intr_driver'에 대한 Write를 수행한다.

```
/* write() Handler for this driver */
static int intr_driver_write(
    struct file *file, const char __user *buf, size_t count, loff_t *f_pos
) {
    printk("[INTR_DRIVER] %s called\n", __func__);

    /* Put the user thread on a sleep */
    printk("[INTR_DRIVER] Make user thread(java thread) sleep\n");
    interruptible_sleep_on(&intrq);

    return 0;
}
```

◆ Write를 시도한 Java Thread는 위와 같은 intr_driver의 write Routine에 의해 자동으로 Sleep하게 된다.

```
/* Interrupt handler for 'back' button, it will trigger the sleeping java thread */
static irqreturn_t _back_button_handler(int irq, void* dev_id) {
    int ret;

    /* Make and schedule the bottom half (printks) */
    work = (Work *)kmalloc(sizeof(Work), GFP_KERNEL);
    if (work) {
        INIT_WORK((struct work_struct *)work, work_function);
        work->button = BACK;
        ret = queue_work(workq, (struct work_struct *)work);
    }

    /* Wake up the sleeping thread (which is a java thread in here) */
    __wake_up(&intrq, 1, 1, NULL);

    return IRQ_HANDLED;
}
```

◆ 한편, intr_driver는 위와 같은 Hander를 Back Button 입력 GPIO에 대해 Interrupt Handler로 등록한다. 즉, Back Button 클릭 시 위와 같은 함수가 동작하는 것이다. 'intrq' Queue에서 Sleep(Wait)중인 Thread를 깨우고 있음을 알 수 있다.

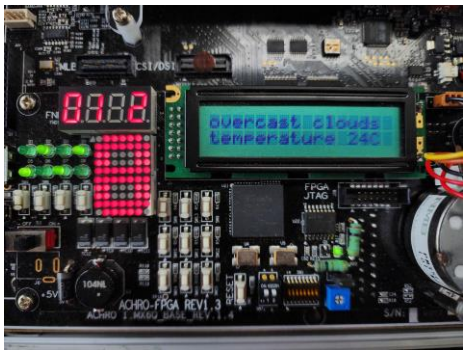
- 참고로 Bottom Half에선 Logging Print문들이 수행된다.

```java
private void toggleMusic() {
    isMusicOn = !isMusicOn;
    // Control the music playback
    if (isMusicOn) {
        myService.startMusic();
        Log.d("TurnOnOffThread", "Music turned on");
    } else {
        myService.stopMusic();
        Log.d("TurnOnOffThread", "Music turned off");
    }
}
```

```java
public void startMusic() {
    mp.start();
}

public void stopMusic() {
    mp.pause();
}
```

◆ 깨어난 Java Thread는 위(좌측)와 같은 동작을 수행해 bindService와 소통하여 음악 ON/OFF를 제어한다.

◆ 위의 우측은 bindService의 일부 루틴을 보여주며, **이러한 루틴들이 MainActivity와 Java Thread에서 불려 Service를 조종**할 수 있다.

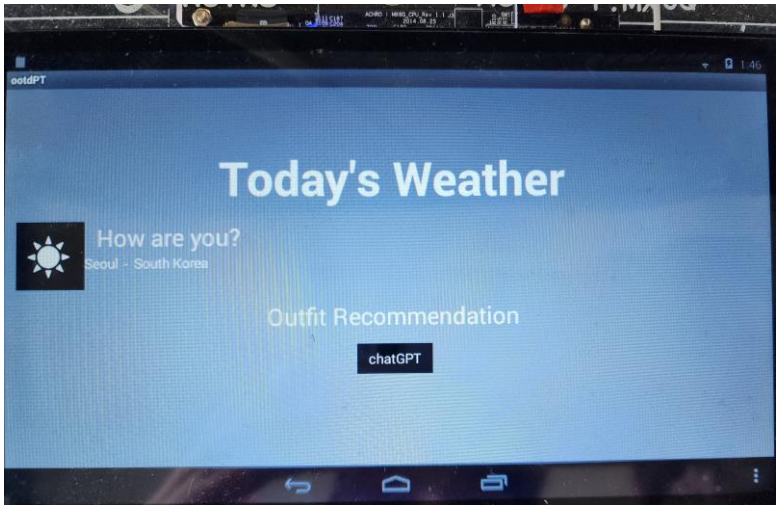


I/O Devices

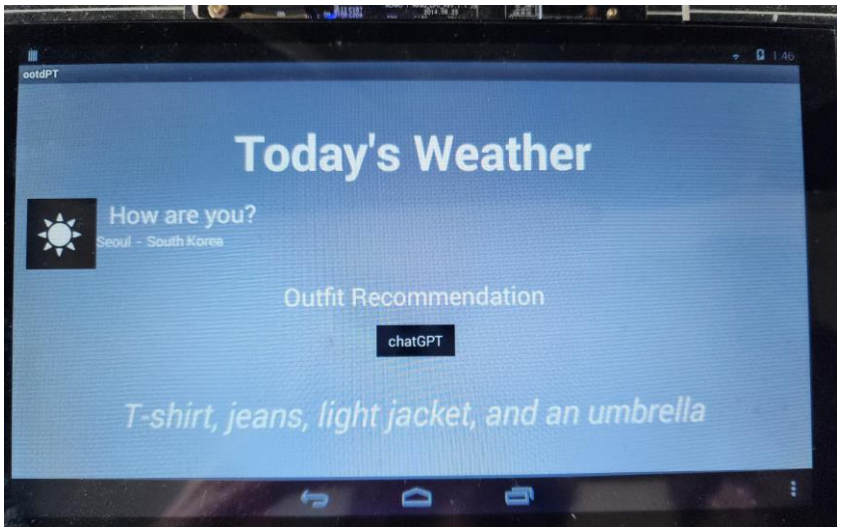
Back Button

Result

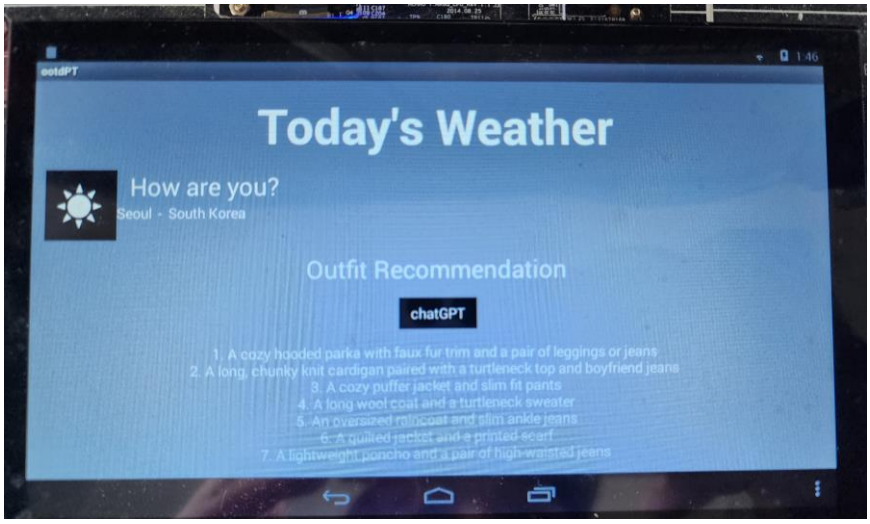
- 위와 같은 과정을 거쳐 만들어진 ootdPT는 다음과 같다.



- App을 시작한 직후 초기화면으로, 아직 openweathermap API의 Callback이 이뤄지기 전이다. 보통 이 Callback은 1초 내로 이루어진다.



- 방안1에 따른 'In-Device Outfit Recommendation Engine'의 결과물이 출력된다. 이때, 이전 장의 'I/O Devices' 사진과 같이 I/O Device 동작도 이뤄진다. 날씨 요약 정보 및 기온, 습도, 날짜 등이 출력되고 있음을 주목하자(날짜의 경우 현재 Device의 System 설정이 1월 2일로 되어 있어 저렇게 출력. 변경 가능).



- 이때, 'chatGPT'라는 버튼을 클릭하면 Remote Server로부터 받은 ChatGPT의 옷차림 추천 결과물도 확인할 수 있다(Text File이 생성되어 있지 않은 시점에 클릭한다면 반응하지 않음. 하지만 Server가 켜져만 있다면 최대 5초 내에 Text File이 생성된다).

- 이렇게 ootdPT의 개발 과정 및 결과물을 모두 확인하였다. **실제로 근 시일 내에 이러한 서비스가 개발될 것으로 예상**되므로 이번 프로젝트 간에 본인이 고민했던 사항이 실제 업계에선 어떤 형식으로 개발할지 기대된다.

강의 내 개발 목록

1) Module Programming (Device Drivers)
2) Kernel Timer and Interrupt Handling
3) GPIO and I/O Device Handling
4) Top and Bottom Half
5) JNI(Java Native Interface)
6) Activity Service, Java Thread(Binder IPC)

강의 외 개발 목록

1) API Calling Mechanism (HTTP Connection)
2) Intermediate Server Approach

최종 소감

이번 임베디드시스템소프트웨어 강의를 통해 그간 항상 당연시 여겨오고 자세히는 확인해보지 않았던 Kernel 내부 동작 및 Android Architecture를 자세히 공부할 수 있어서 개인적으로 매우 뜻 깊었습니다. 마지막 최종 프로젝트 수행 간에 개인적인 일정(캡스톤디자인1 및 논문 작업 등)이 과다하게 몰려 아쉽게도 Android 또는 Kernel 그 자체의 변형을 시도해보지 못한 것은 아쉽지만, 그럼에도 불구하고 수업 시간에 배운 내용을 충분히 활용하며 실생활에 유용할 실용적 Service를 개발해볼 수 있었기에 후회는 없습니다. 지금까지 열정적으로 좋은 강의를 해주신 박성용 교수님과, 실습 및 과제 간에 많은 조언 및 도움을 준 조교님께 큰 감사를 드립니다. 고생하셨습니다, 그리고 감사합니다! ☺