멘토링 보고서

AOSP 조

조원 : 김재현, 이송무, 임진현

지도교수: 조은선

Document Revision History

Rev#	Date	Affected Section	Author
1	2020/05/20 ~ 05/21		김재현
2	2020/05/21 ~ 05/22		이송무
3	2020/05/22 ~ 05/23		임진현

보고서 작성은 구글 드라이브 실시간 온라인 문서 작성을 이용하여 진행을 하였다. 작성하는 시간대를 사전에 나눠서 보고서를 구성하였다.

Table of Contents

1. Introduction

1.1. Objective

이 문서는 멘토링 결과 및 설문조사 분석을 정한 보고서이다. 2월 9일 부터 프로젝트는 시작을 했으며 2월 마지막 주를 시작으로 주 1회 미팅을 가졌다. 미팅은 Google hangout으로 화상회의로 진행되었으며, 총 13번 미팅을 가졌다. 주 마다 각 팀원은 AOSP (Android Open Source Project)와 그에 관한 프레임워크를 공부를 하고 진행을 하였다. 팀원 전부 공부를 한 내용을 깃허브에 업로드 하였으며, 주 마다 업로드 한 내용을 발표하였으며, 그에 대해 교수님께서 피드백을 하셨다. 안드로이드는 자바 프레임워크 단에서부터 네이티브 단까지 이루어져있으며 이번 프로젝트는 Android GO Edition 을 기점으로 할 예정이다.

Java Core Platform Framework (WindowManger) → Native Platform (Surface Flinger) 이다. 아래 주소는 각 팀원들의 깃허브 블로그 주소이다.

김재현 : https://keelim.github.io/AOSP/

이송무 : https://songmoolee.github.io

임진현 : https://nivkhdif.github.io

멘토링 보고서

지도 교수님과의 회의 내용

주 1회 지도 교수님과의 팀 회의를 가졌으며 Google hangout을 통하여 진행을 하였다.

LG전자 멘토님과 함께 학생들이 계획에 적어준 1~4에 대해 논의했습니다.

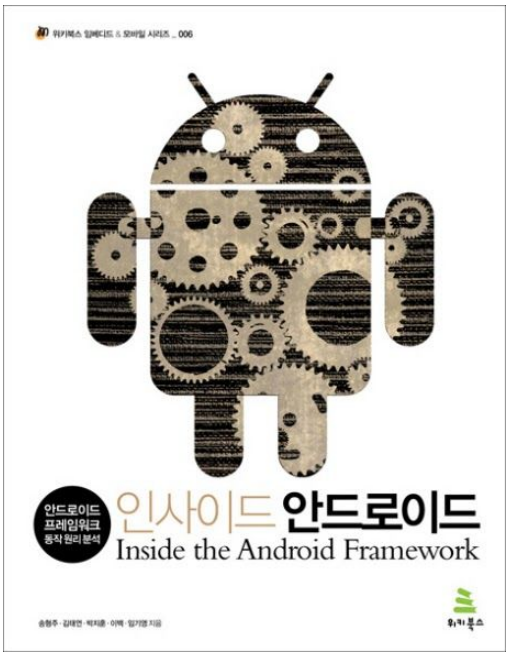
1. 커스텀 롬, 커널 사례 연구=> 사례 연구에서 끝나지않고, 커스텀 빌드 후 코드를 수정하여 개선한다면 괜찮을 것 같습니다.

2 Treble에 대한 멘토님의 의견입니다.
=> 아직 완벽한 기능이 아니라고 생각됩니다. 구글에서 처음에 시작할때는 이론적으로는 탄탄했으나 실제로 100% 적용되어 있진 않다고 알고 있습니다. 그래서 Treble이 어떤 목표로 만들어졌고 어떤식으로 hardware dependency를 끊어 낼수 있는가? 와 같은 이론을 연구해보는것까지는 괜찮을텐데 뭔가 눈에 보이는 결과물을 만들어서 실행해본다거나 하는건 어려울것 같습니다.

3. 안드로이드 OS를 적용함에 있어서 Pixel과 같은 AOSP가 적용되는 기기와는 별도로 각 제조사별 AOSP 커스텀에 관한 방법 연구에 대한 멘토님 의견
=> 첫번째 주제에 연결되는 내용일텐데, 순수 AOSP에 제조사별 커스텀이 어떤식으로 들어가느냐를 비교하는건 어렵습니다. 제조사에서 만드는 코드들은 Open source license외에는 모두 대외비 이기 때문에 이를 외부에서 확인하기에는 어려움이 있습니다. 다만 연구 자체에 의미를 둔다면, 동작에 대한 dump나 로그 등으로 어느정도의 구조 파악은 해볼 수 있습니다. 타겟에 실제 올려보고 동작하기에는 제한이 많습니다.

4. Android Go 대한 멘토님 의견
=> 일단 Framework에서 어느정도 저사양 단말을 위해 기능 제한이나 feature off 등을 하는데, 이를 외부에서 따로 커스텀 빌드 한다거나 하는건 조금 제약이 있을것 같은데요. 따라서 만약 이쪽을 연구해보고 싶다 하면, Android Go를 위한 적용 기술을 일반 AOSP에 적용해본다거나 하는 방법은 해볼만 할 것 같습니다.

주 1회 회의를 바탕으로 각 블로그의 WindowMangaer 와 SurfaceFlinger 관련을 하여 Github Pgae 를 활용하여 포스팅을 진행하였다.



프레임워크 단의 내용을 확인 하기 위하여 SDK (Software Development Kit) 수준인 PDK(Platform Development Kit) 수준으로 진행을 하였다.

WindowManager 코드 본격 분석 2

개념 보충 프로젝트를 진행을 하면서 개념 보충 해야 하는 것들을 진행을 해보기로 하였다. WindowMangaer -> Surface Flinger 위에 위치하며, 기기화면에 그릴 내용을 Surface Flinger 로 전달 Core Platform Service 어플리케이션과는 직접 상호작용은 하지 않지만 안드로이드 프레임워크가 종작하는데 필수적인 서비스 Activity Manger, Package Manger 우연찮게 겹친다. 안드로이드 플랫폼 이해 용어 정리...

May 19, 2020

in Aosp

WindowManager 코드 본격 분석

WindowManager 대략적인 설명 각 WindowManager 인스턴스는 특정 디스플레이에 바인딩됩니다. 다른 디스플레이에 대한 WindowManager를 얻으려면 Context # createDisplayContext를 사용하여 해당 디스플레이에 대한 컨텍스트를 얻은 다음 Context.getSystemService (Context.WINDOW_SERVICE)를 사용하여 WindowManager를 가져옴 1 //wm 1 Window Manager 이번 분석에서 다루고자 하는 부분은 이 3가지의 클래스이다. WindowManagerPolicyConstant 클래스도 존재를 한다. 이 과정을 Window Manger에서...

May 12, 2020

in Aosp

액티비티 매니저 분석

액티비티 매니저 서비스 액티비티 매니저 서비스: 자바 시스템 서비스이며, 안드로이드 애플리케이션 컴포넌트(액티비티, 서비스, 브로드캐스트 리시버 등)를 생성하고, 이들의 생명주기를 관리한다 애플리케이션 서비스의 실행을 요청한 애플리케이션 프로세스와 액티비티 매니저 서비스가 어떻게 상호작용하는가 Remot Service Controller 예제 애플리케이션: 'Start Service' 버튼을 누르면 Remote Servcie가 시작되는 프로그램 애플리케이션에서 서비스를 실행하면, 액티비티 매니저 서비스가...

April 20, 2020

in Aosp

Recent Posts

스터디 10주차
10 minute read
AIDL 관련

스터디 9주차
5 minute read
WindowManager

스터디 8주차
2 minute read
자바 서비스 프레임워크

스터디 7주차-2
4 minute read
바인더 코드 분석

Recent Posts

윈도우 매니저

2 minute read

SurfaceFlinger:: 여러 개의 Surface를 하나의 Surface로 만들고

바인더 관련

1 minute read

바인더

Jni관련

2 minute read

SDK와 NDK, 그리고 PDK

기본카메라앱 생성의 흐름

1 minute read

카메라생성흐름

산업체 멘토님과의 회의 내용

Android GO 부분에서 연관이 있을것 같은 부분은 특별히 생각하고 있는 부분이 있어서일까요?
WindowManager쪽은 실제로 화면에 보이는 부분과 밀접한 곳입니다. 그래서 코드가 조금 지져분 하기도 합니다.
다만 현재 테스트할수 있는 환경이 실제도 단말에 올려볼수 있지 않다면 에뮬레이터 등을 이용해야 할텐데 성능에 영향을 받는 부분이라 감안하셔야 할 것 같습니다.

그리고 추가로 찾아볼만한 자료가 필요하다면, Developer site나
http://cafe369.daum.net/_c21_/home?grpid=1MWA2 이쪽 카페도 많은 자료가 있습니다.
조금 오래되긴 했지만 참고해볼만합니다.

보고서 작성하시고 하면서 궁금한점이나 프로젝트 진행하시면서 연락 필요하면 메일 주세요

그럼 잘 부탁드립니다.

감사합니다.

산업체 연구원 멘토님과 메일을 주고받음으로써, 앞으로 좀더 Window Manager 단에서 코드를 분석하고 진행하려고 한다

현재 프로젝트 진행 상태

- 텔레스코핑 패턴 및 레거시 코드들을 확인
- WindowManagerService의 addWindow 클래스에서 다중 if else문으로 인한 성능 저하 확인 (추가 해야되나? switch 문에 전환 혹은 해싱으로 인한 처리 확인)
- WindowManagerService addWindow 중복 return 으로 인한 처리 확인
- 객체 단위에서 synchronized 를 통한 성능 저하 확인
- 잘못된 오류 처리 확인

코드리뷰 부분

```
if (token == null) { //같은 리턴을 반복한다. -> 개선을 할 수 있을 것 같다.
    if (rootType >= FIRST_APPLICATION_WINDOW && rootType <= LAST_APPLICATION_WINDOW) {
        Slog.w(TAG_WM, "Attempted to add application window with unknown token "
              + attrs.token + ".  Aborting.");
        return WindowManagerGlobal.ADD_BAD_APP_TOKEN;
    }
    if (rootType == TYPE_INPUT_METHOD) {
        Slog.w(TAG_WM, "Attempted to add input method window with unknown token "
              + attrs.token + ".  Aborting.");
        return WindowManagerGlobal.ADD_BAD_APP_TOKEN;
    }
    if (rootType == TYPE_VOICE_INTERACTION) {
        Slog.w(TAG_WM, "Attempted to add voice interaction window with unknown token "
              + attrs.token + ".  Aborting.");
        return WindowManagerGlobal.ADD_BAD_APP_TOKEN;
    }
    if (rootType == TYPE_WALLPAPER) {
        Slog.w(TAG_WM, "Attempted to add wallpaper window with unknown token "
              + attrs.token + ".  Aborting.");
        return WindowManagerGlobal.ADD_BAD_APP_TOKEN;
    }
    if (rootType == TYPE_DREAM) {
        Slog.w(TAG_WM, "Attempted to add Dream window with unknown token "
              + attrs.token + ".  Aborting.");
        return WindowManagerGlobal.ADD_BAD_APP_TOKEN;
    }
    if (rootType == TYPE_QS_DIALOG) {
        Slog.w(TAG_WM, "Attempted to add QS dialog window with unknown token "
              + attrs.token + ".  Aborting.");
```

안드로이드 release 10을 기반으로 분석 시작

설문조사 분석

개선된 안드로이드에서 마음에 들었던 점이 있다면 어느 점에서 장점을 느꼈는가?

: 전보다 빠른 속도 (4명), 코드 가독성 향상 (1명)

안드로이드에서 신뢰할 수 없는 기능이나 단점이 있다면 어떤 것이며 그 이유는 무엇인가?

: 하드웨어의 성능만 믿고 통으로 짠 코드가 많다, 가용램이 적다, 레거시 코드가 많다.

개선된 안드로이드에서 추가된 기능이 필요 없다고 생각하는 부분이 있는가?

: try / catch 문을 재확인 할 필요성이 있다, 오류 확인 부분에 있어 관습적인 코드 일수도 있다.

안드로이드에서 개선의 여지가 있는 부분이 있다면 어떤 것이며 그 이유는 무엇인가?

: 안드로이드 JNI를 사용하기 때문에 관습적인 코드가 있지 않을까?

텔레스코핑 패턴 말고도 다른 레거시 패턴들이 있지 않을까?

C코드를 쓰는 것이 더 빠르지 않을까?, 배터리 사용을 줄일 수 있을까?

어느 방면에서 성능이 향상되었다고 생각하는가?

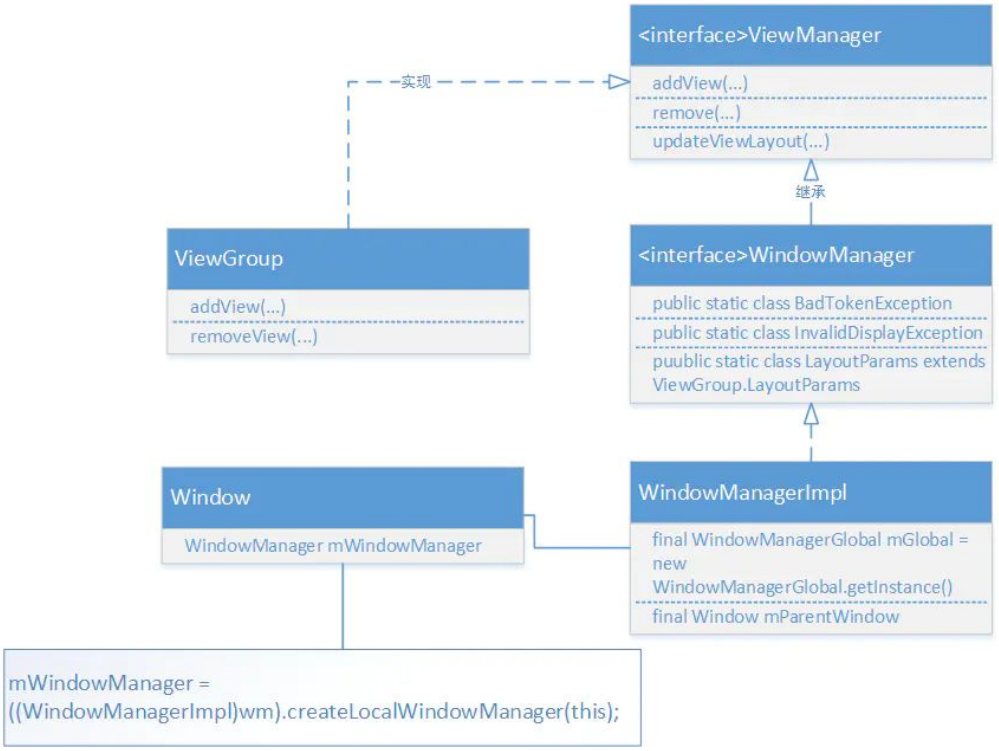
: 대부분 실행 속도(4명), 코드의 가독성 및 앞으로의 코드 개발 여부

지난 설문 분석

지난 설문조사를 분석을 하였을 때 Telescoping 패턴, try-catch문 등과 관련하여 진행을 하면서, 네이티브(하드웨어) 단에서 까지 진행을 하는 것으로 프로젝트를 심화를 시켰으며 Surface Flinger 레벨까지의 분석을 시작하고 있다.

```
SurfaceFlinger::SurfaceFlinger(Factory& factory) : SurfaceFlinger(factory, SkipInitialization) {
    ALOGI("SurfaceFlinger is starting");

    hasSyncFramework = running_without_sync_framework(true);

    dispSyncPresentTimeOffset = present_time_offset_from_vsync_ns(0);

    useHwcForRgbToYuv = force_hwc_copy_for_virtual_displays(false);

    maxVirtualDisplaySize = max_virtual_display_dimension(0);

    // Vr flinger is only enabled on Daydream ready devices.
    useVrFlinger = use_vr_flinger(false);

    maxFrameBufferAcquiredBuffers = max_frame_buffer_acquired_buffers(2);

    hasWideColorDisplay = has_wide_color_display(false);

    useColorManagement = use_color_management(false);
```

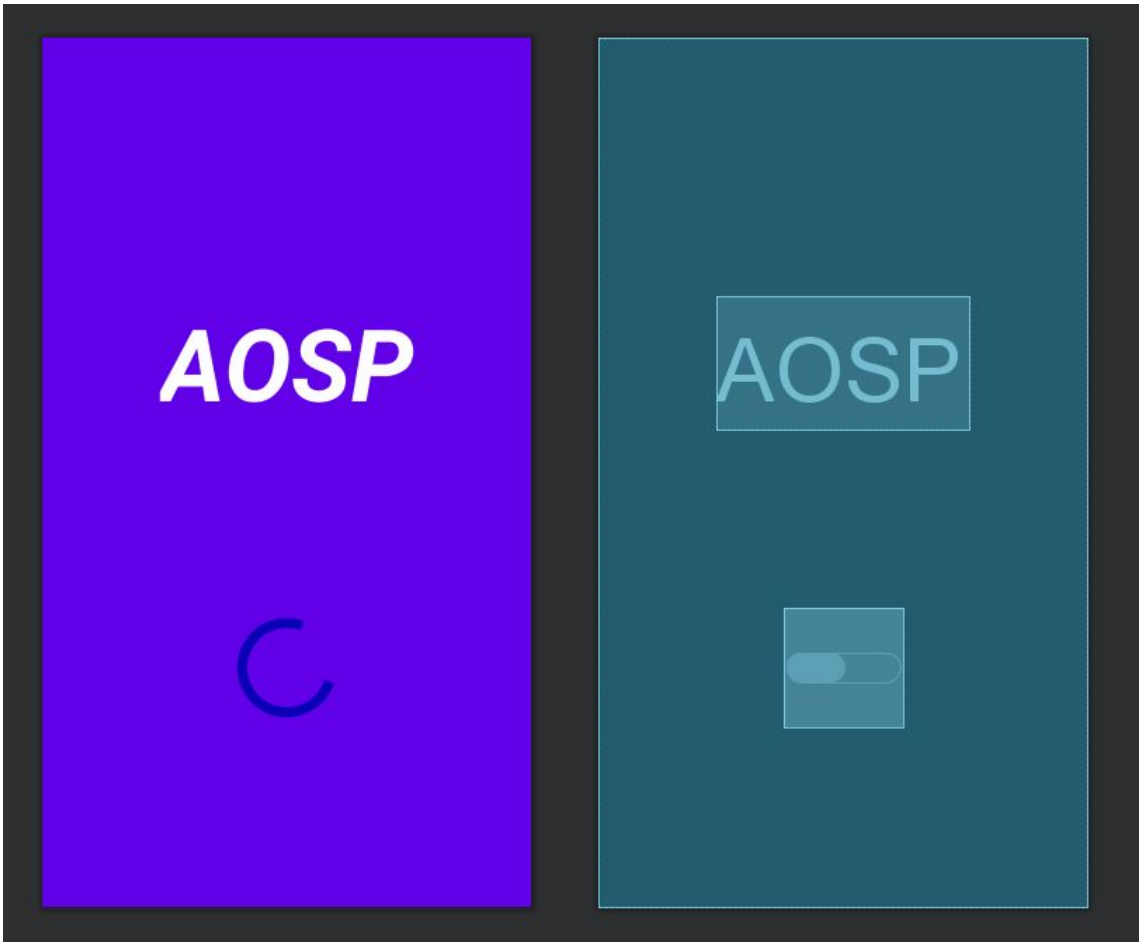
마무리



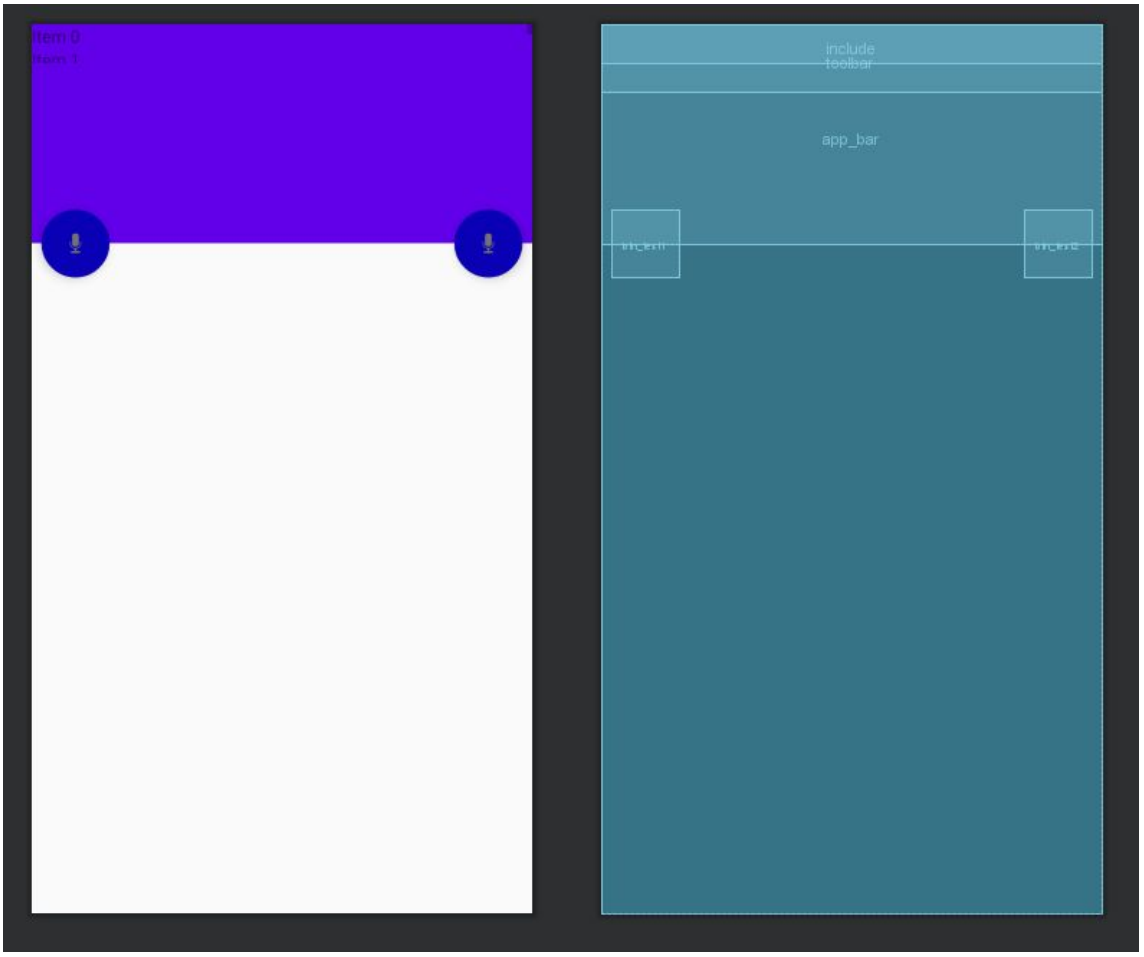
우선 Java level 에서 프레임워크 성능을 개선하기 위하여 메소드 단위 분석을 마무리 한 후 부족한 부분을 찾아 개선을 하는 것이 목표이다.

프로토타이핑

이번 프로젝트에 관하여 프로토타이핑에 관한 설명이다.



테스팅 데모 1



테스팅 데모 2

안드로이드 Activity 실행에 있어서 SplashActivity → MainActivity (MainAdapter, MainModel) 등에 프로토타이핑 생성 및 실행 파일 구성 각 Floating Button 으로 Test1, Test2 진행