정보통신산업진흥원 부설 SW공학센터

SW아키텍처 참조모델

[안드로이드를 기반으로한 품질 개선 아키텍처 참조모델]

[Version 1.0 20121230]

SW공학센터

SW공학기술팀

SW아키텍처 실무자 포럼 모바일 분과

SEC-2012-RM004

본 문서는 국내 기업의 소프트웨어 품질 및 생산성 향상을 지원하기 위하여 작성되었습니다.

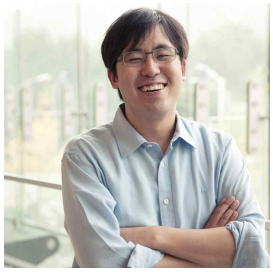
본 문서는
지식경제부 산하
정보통신산업진흥원 부설
SW공학센터
SW아키텍처 실무자 포럼
에서 작성되었습니다.

SW공학센터는 SW제품 생산능력 향상, SW공학기술 산업현장 적용 등을 위해 대학 및 전문 연구기관과 기업 현장을 연결하는 중심 허브가 되어 SW개발 중소기업들에게 전문 컨설팅을 제공하고 있습니다. 이 같은 역할을 충실히 수행하기 위해 산업과 기업의 SW공학기술 관련 요구사항에 전문적이고 신속한 대응을 할 수 있는 핵심 기능 연구와 SW개발 프로젝트의 성공 여부와 문제점을 미리 예측하여 SW품질과 생산성, 제품결함 등을 총체적으로 진단 할 수 있는 SW공학 컨설팅 등도 추진하고 있습니다. 이와 더불어 SW품질과 생산성, 비용 등을 체계적으로 추적 평가할 수 있는 데이터 수집체계도 강화하여 SW기업들이 이를 손쉽게 활용하게 함으로써 전체적인 국가 SW품질을 향상시키는 업무도 수행중입니다.

본 지침에 대한 더 많은 정보와 SW공학에 대한 추가 정보를 얻고 싶다면, SW공학센터 홈페이지(www.software.kr)를 방문하여 주십시오.

담당자 강승준 책임 [ksj@nipa.kr, 02-2132-1344]

분과원 소개 및 도움 주신 분

이름	소개	경력
 손영수 분과장 NHNNEXT 교수 indigoguru@gmail.com	소프트웨어 공학과 아키텍팅을 국내 개발자들에게 널리 알리기 위해 노력하고 있으며, NHNNEXT 에서 소프트웨어 아키텍팅과 모바일을 가르키고 있다. 국내 최초의 패턴 전문가 Hillside 그룹의 이사회 위원이며, 아시아 패턴 학회인 AsianPLoP 의 공동 의장이다. 지식 공유를 위해 세미나, 워크숍, 기고 등 왕성한 활동을 하고 있으며, 여러 스타트업 회사들을 지원하고 있다. 요즘 후배들을 양성하고 있으며, 더 큰 성장을 위해 수련을 하고 있다.	- 소프트웨어 마에스트로 멘토를 포함한 다양한 멘토링 경험과 다수의 공모전 수상. - 한국 공개 SW 포럼 인력양성 분과장 - AsianPLoP 공동의장. - PLoP 이사회 Hillside Group Board Member - PLoP (소프트웨어 패턴학회) 인증 소프트웨어 패턴 저자 활동 및 다수의 패턴 발표. - 소프트웨어 공학 커뮤니티 EVA 10 년간 운영. - "Hybrid Push/Pull 소프트웨어 업데이트"등 미국 특허 3 건/ 국내 특허 3 건 등록. - "아키텍트가 알아야 할 97 가지" 다수의 소프트웨어 관련 서적 집필 및 번역.
 김영호 위원 말랑 스튜디오 CEO	우리는 누구를 위해 소프트웨어를 개발하는 것인가? 라는 질문에 대한 답을 찾고자 회사와 학교를 뛰쳐나와 창업을 선택하였다. 또한 좋은 사람이 좋은 서비스를 만들 수 있다는 목표로 매일 같이 좋은 사람을 찾아다니는 중이다. 현재 (주)말랑스튜디오의 대표를 맡고 있으며, 사람들의 모바일 라이프스타일 속에 숨어 있는 진주를 캐내기 위해 인고의 노력을 하고 있다.	- 삼성 소프트웨어 멤버십 - 삼성서울병원 PACS System 설계/개발 - SBS 인터넷 라디오 "고릴라 2.0" 개발 - 소프트웨어 마에스트로(지식경제부 장관상) - 제 3 회 코리아 모바일 어워드 최우수상 - SK T-Store Award 디자인혁신상 - 글로벌 K 스타트업 우수상 및 Google Choice - Global Super Apps 최우수상 - 2 회 청년기업가정신대회 인기상/특별상 - 서강대학교, 강원대학교 등 다수 특강
 진성주 위원 KTH PD	항상 공유하며 성장하고픈 개발자이다. 모바일관련된 개발을 주욱 이어오고 있고 오픈소스 활동을 하며 글로벌 엔지니어가 되려고 수련을 하고 있다. 현재는 모바일 백엔드 서비스 오픈소스 프로젝트인 Apigee usergrid 프로젝트에서 컨트리뷰터로 활동하고 있다.	- (저) 안드로이드 프로그래밍 : 제대로 된 안드로이드 앱 개발을 위한 - SW 아키텍트 실무자 포럼 모바일 분과 위원, NIPA SW 공학센터 - 2012 공개 SW(OSS) 개발자대회 멘토, 공개소프트웨어협회 - 11, 12 회 JCO 발표, 2012 H3 발표, OpenTechnet 등 강연활동
 강진석 위원 프리랜서	아직도 배울것이 많다고 느끼고 배워나가고 있는 개발자이다. 바다, 안드로이드와 같은 모바일 관련 개발을 해왔으며 현재 네트워크, 서버등에 관심을 갖고 공부하고 개발하고 있다.	- 삼성 소프트웨어 멤버십 - 삼성전자 bada 모바일 앱 공모전 1 등 - 매일경제 안드로이드 앱 공모전 최우수상 - SK T-Store Award 디자인혁신상 - 글로벌 K 스타트업 우수상 및 Google Choice 수상 - Global Super Apps 최우수상 - 제 2 회 청년기업가정신대회 인기상 및 특별상 수상

 신재명 위원 KAIST 대학원	개발을 통해 실제 생활에서 불편한것을 해결해나가는것을 좋아하는 개발자이다. 모바일 관련 개발을 진행해 왔으며 관심분야는 Mobile computing, Pervasive computing 등이 있다. 현재는 카이스트 대학원 석사과정에서 AI 와 Mobile computing 을 공부하고 있다.	- 대표앱 / Sleep If U Can : 지정된 사진을 찍어야지만 꺼지는 알람, 현재 약 160 개국에서 사용되고 있음 -소프트웨어 마에스트로 2 기 최종인증자 (지경부장관상) - 2012 한국정보과학회 우수논문상 - KT Go to Global 앱경진대회 특별상 - 제 6 회 공개 SW 개발자대회 오프닝 강사 - 삼성전자 휴대폰사업부 스마트포럼 강의
 오유환 위원 LG 전자 연구원	모든 사람들이 필요하다고 느낄 수 있는 프로그램을 만들고 싶어한다. 다양한 분야에서 경험하고 싶어 하며, 경험을 하면서 배우고 느낀 것을 함께 나누기 위해 노력하고 있다.	- 소프트웨어 마에스트로 2 기 최종인증자 (지경부장관상) - 공개 SW 개발자대회 그랜드오프닝, 커뮤니티 데이 세미나 강사 - Micro Software 잡지 다수 기고 - 페이스북 프레임워크 fhalo 커미터 - 아름다운 약속 기부앱 개발
 강미경 위원 소프트웨어 마에스트로 멘티	꿈을 향해 도전하고, 배움을 통해 성장해 나가는 여성 개발자이다. 모바일 관련 개발을 시작으로 소프트웨어 아키텍쳐, 네트워크 등 다양한 분야에 관심을 갖고 공부하고 있다.	- 소프트웨어 마에스트로 2 기 - 삼성 소프트웨어 멤버십 - Micro Software 잡지기고 (5~7 월) - 페이스북 프레임워크 fhalo 커미터 - 아름다운 약속 기부앱 개발"
 김민정 위원 투이컨설팅 컨설턴트	기술제민의 사명으로 정보 기술 선진화를 통해 고객의 비즈니스 가치를 실현하고 궁극적으로는 고객의 고객을 위한 서비스를 제공할 수 있도록 노력하고 있다.	- 소프트웨어 공학 커뮤니티 EVA 팀 - 패턴 지향 소프트웨어 아키텍처 1 감수 - 아키텍트가 알아야 할 97 가지 번역 - 브랜드 어플리케이션 코드캠프 2 기

목차

1. 개요 및 배경

1-1. 안드로이드(Android) 플랫폼

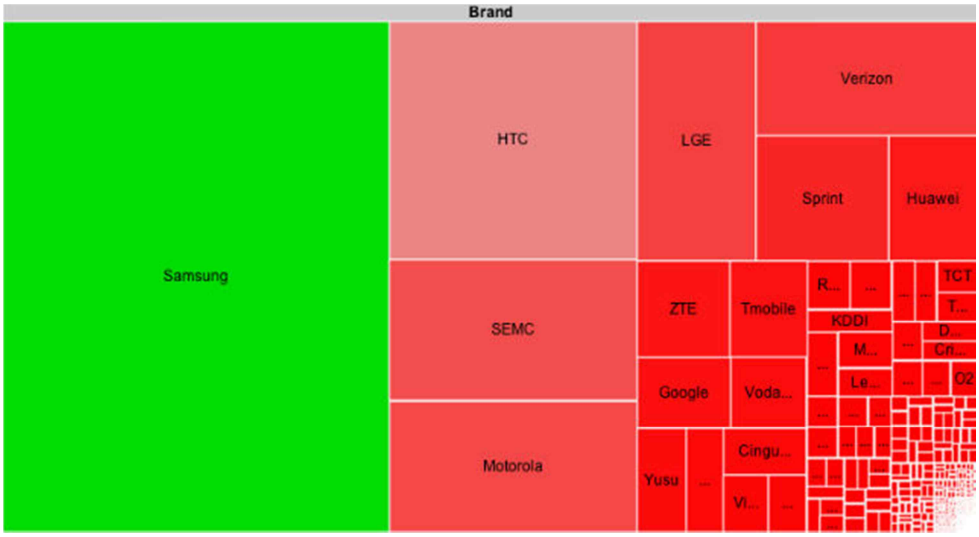
안드로이드(Android)는 휴대 전화를 비롯한 휴대용 장치를 위한 운영 체제와 미들웨어, 사용자 인터페이스 그리고 표준 응용 프로그램(웹 브라우저, 이메일 클라이언트, 단문 메시지 서비스(SMS), 멀티미디어 메시지 서비스(MMS)등)을 포함하고 있는 소프트웨어 스택이자 모바일 운영 체제이다.
안드로이드는 개발자들이 자바 언어로 응용 프로그램을 작성할 수 있게 되어있으며, 컴파일된 바이트코드를 구동할 수 있는 런타임 환경을 제공한다. 또한 안드로이드 소프트웨어 개발 키트(SDK: Software Development Kit)를 통해 응용 프로그램을 개발하기 위해 필요한 각종 도구들과 응용 프로그램 프로그래밍 인터페이스(API)를 제공한다.

자세한 안드로이드에 대한 상세 설명은 아래 링크를 참조하도록 한다.
http://developer.android.com/index.html

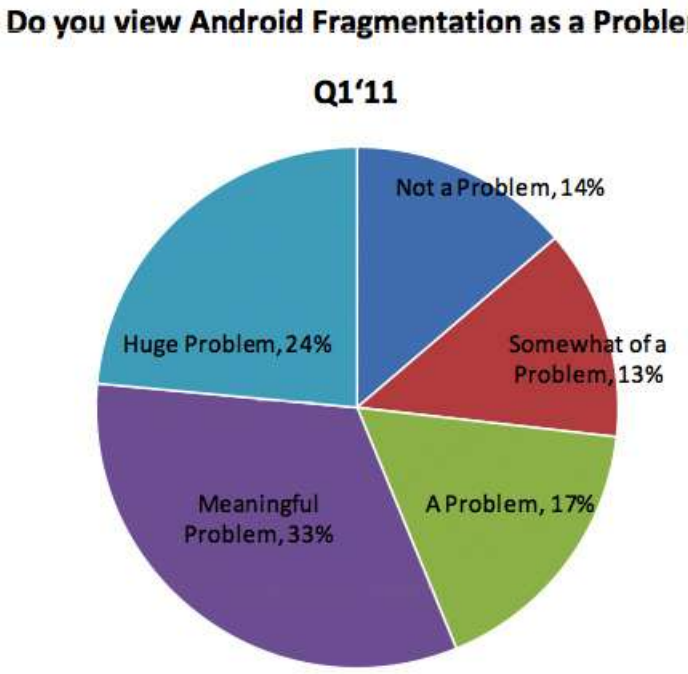
1-2. 모바일 오픈 아키텍쳐 레퍼런스 추진 배경

1-2-1. 안드로이드 OS 의 파편화

안드로이드 OS 의 버전이 빠르게 업데이트 되면서 그에 따른 소프트웨어의 크기와 복잡도가 상상을 불허하게 커져갔고, 또한 제조사별로 수 많은 디바이스를 내놓으면서 그에 따른 안드로이드 OS 의 파편화가 매우 심해지기 시작했다. 그리고 잦은 업그레이드로 인해 불과 몇년사이에 5 종이상의 안드로이드 버젼이 생겨나면서, 개발자들은 하나의 어플리케이션을 개발하기 위해 적어도 3 종 이상의 상호 호환이 장담되지 않는 안드로이드 버젼을 맞춰야하는 불편함에 직면하게 되었다.
즉, 하나의 feature 를 개발하고도 몇 개이상의 버젼에서 정상 작동 여부에 대해서 테스트를 해야 하고, 이러한 테스트 활동은 그대로 개발기간에도 반영이 되어서 시장의 요구에 바로 대응해야 하는 부분에 있어서 매우 불편함을 초래한다. 이는 하위 버전과의 완벽한 호환성을 가져가는 iOS 에 비해서 안드로이드가 가지고 있는 최대의 약점이기도 하다.

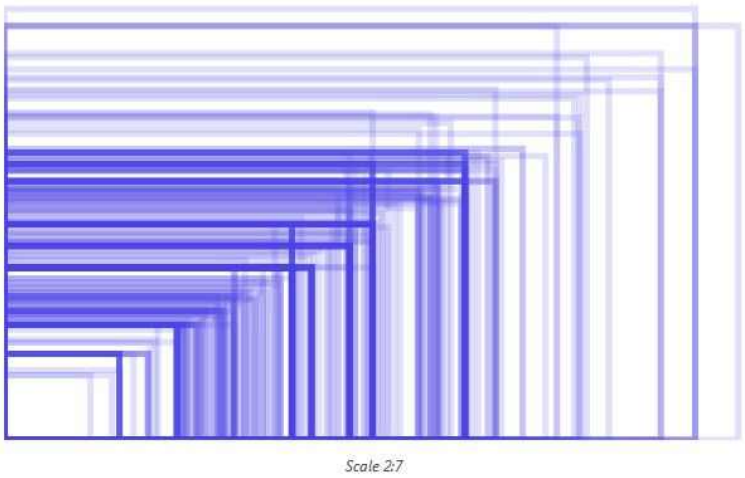


[그림 1. 안드로이드 디바이스 제조사에 따른 파편화 현상]



[그림 2. 안드로이드 플랫폼 파편화 불편함 설문 조사]

위의 그래프는 안드로이드 파편화 문제로 불편함을 겪었는지에 대해 설문조사한 결과로써 문제를 겪었다는 사용자들이 14%를 제외한 대부분으로, 많은 어플리케이션을 활용하지 않는 라이트 유저들을 제외하고 어느 정도 활용법을 익힌 안드로이드 유저들은 불편함을 겪었다는 결과로 해석될 수 있다.



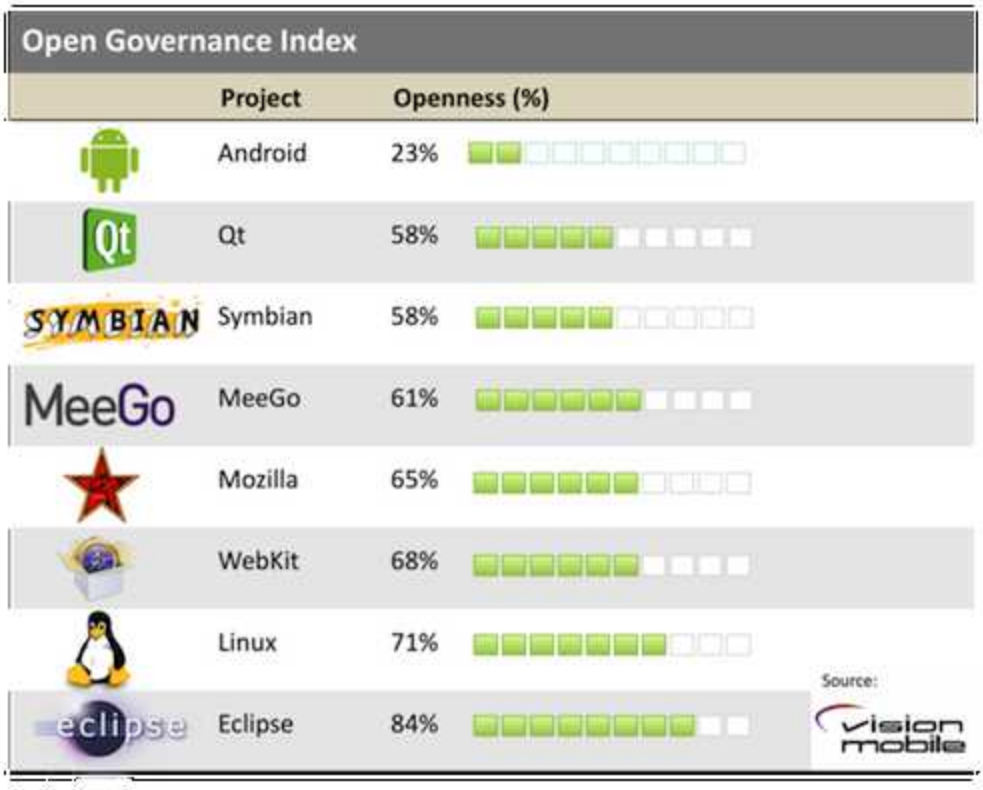
[그림 3. 안드로이드 기기들의 해상도]

그림 3 은 안드로이드 기기들의 해상도를 모두 표시한 것으로써 일정한 화면비를 따르지 않음은 물론이거니와 크기 또한 제각각으로, 가장 표준화된 해상도인 800x480 이나 HD 해상도를 제외한 다른 폰에서는 다소의 호환성 문제가 발생한다. 또한 안드로이드에서는 iOS 에서와 같은 해상도 증폭기능도 사용 할 수 없기에 어플리케이션의 빈약함이 가장 큰 약점으로 지적되고 있는 안드로이드 태블릿의 약점을 더욱 키우는 하나의 요소가 된다.

그러므로, 이와 같이 여러가지 이유 및 현상으로 판단하건데, 안드로이드 OS 에서의 파편화 현상은 개발자, 제조사뿐만 아니라 사용자에게도 심각한 문제로 대두되고 있다.
그에 따라 많은 수의 개발자 및 개발사들은 파편화에 대응할 수 있도록 개발 소스를 매번 수정한 후 본인들의 개발에 적용하고 있다. 이는 개발 생산성이나 효율성에서 심각한 감점 요소로 작용하고 있고, 특히 다양한 사업자 및 사용자를 대응하기 힘든 소규모 회사나 개발자들의 경우에 그러한 현상이 더욱 심각하게 나타난다.

1-2-2. 오픈 소스로써의 개방성 문제: 폐쇄성

2011 년 Vision mobile 에서 보고한『오픈성을 측정하는 새로운 방법: 오픈 거버넌스 지표(A new way of measuring Openness, from Android to WebKit: The Open Governance Index) 』에 보면 안드로이드는 가장 폐쇄적인 구조를 가지는 오픈 소스로 조사되고 있고, 개방성을 의미하는 Open Governance Index 가 23%로써 가장 낮은 점수를 나타 내었다.



[그림 4. Open Governance Index]

즉, 안드로이드는 오픈 소스 소프트웨어 임에도 불구하고 폐쇄성이 강할 뿐만 아니라, 구글(Google)이라는 특정한 회사에 의해서 정책이 결정되고 commit 이 받아들여지는 커뮤니티 구조를 가지고 있다. 따라서, 일반 개발자들이 안드로이드 오픈 소스에 참여해서 새로운 feature 를 추가하기란 여간 어렵고 시간이 소요되는 일이 아닐 수 없다.

더불어 구글의 지속적인 정책 변화로 인해서 안드로이드 OS 의 버전이 올라갈 수록 점점 폐쇄적인 성향이 짙어지고 있다.

1-2-3. 품질 속성 문제: 사용성

안드로이드에서 제공하는 기본 기능을 활용하여 개발하려면 일반적으로 개발자나 사용자가 원하는 기능들을 넣기가 매우 힘들다. 안드로이드는 구글에 의해서 주도되는 오픈 소스 프로젝트인 관계로 대부분의 feature 및 방향들이 구글에 의해서 결정되고 유도된다. 즉, 일반적인 오픈 소스 프로젝트처럼 개발자가 필요해서 제안하는 feature 나 사용성이 높은 기능들은 아무리 좋아도 구글의 방향이나 의도에 부합되지 않으면, 받아들여지지 않을 가능성이 높다.

그로 인해 개발자가 원하거나 시장이 요구하는 기능임에도 불구하고 안드로이드 기본기능에 포함되어 있지 않으면 사실상 구현하는데 어려움이 많고 사용성이 저하된다.

또한, 안드로이드에서 제공하는 기본 기능으로는 다양한 요구 - 기획, 디자이너로부터 -를 충족하기 어려울 뿐 아니라, 다수의 어플리케이션이 경쟁하는 안드로이드 시장에서 차별화를 가지기 위한 요소의 개발에 어려움을 있을 수 있다. 즉, 안드로이드에 많은 경험이 있지 않는 초/중급 개발자들은 다양한 요구사항을 구현하기 위해서는 많은 시행착오를 거쳐야 하는 경우가 많다.

그래서, 실제로 안드로이드에서는 구현하기 어려우나, 개발자들이 편하게 사용할 수 있는 feature 나 기능들을 구현하기 위해서 수많은 오픈 소스 프로젝트가 안드로이드 native 가 아닌 외부 프레임워크나 라이브러리를 통해서 개발되고 제공되고 있다. 외부에서 제공되는 프레임워크나 라이브러리는 안드로이드 native 에서 제공할 수 없는 여러 가지 편의성을 전달함으로써 개발측면의 사용성을 높이고 및 효율화에서 많은 도움을 줄 수 있다.

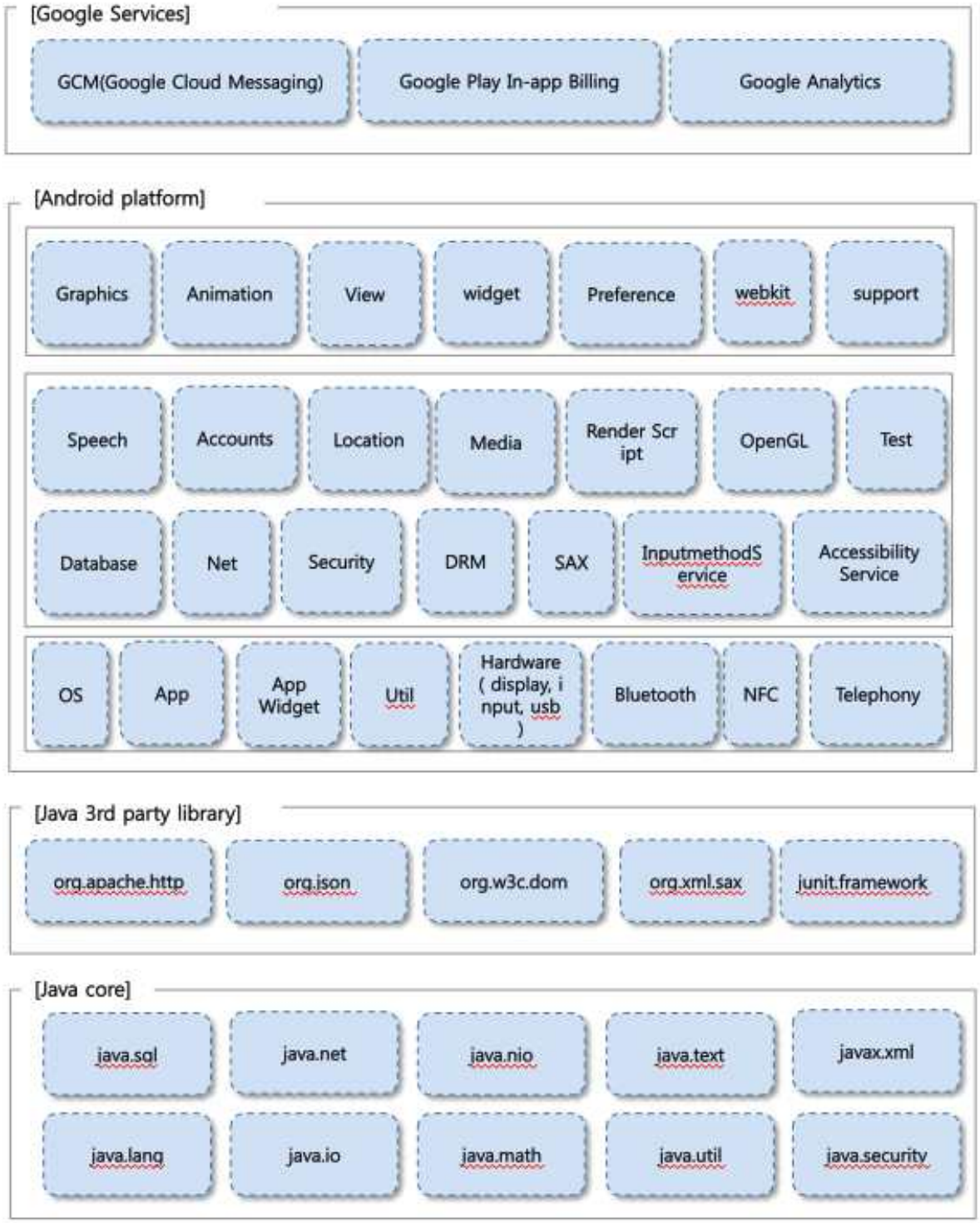
1-2-4. 품질 속성 문제: 유지보수성

안드로이드 OS 를 사용하다 보면, 개발 및 유지보수를 위한 수많은 backend service 들을 편하게 사용할 수 없는 문제가 있다. 실제로 API 가 없을 뿐만 아니라 아예 관련 기초 서비스가 없는 경우가 많다. 예를 들어, 앱 배포 후 유지보수나 서비스 향상을 위한 로그 분석을 위해서 안드로이드에서 제공하는 로그 관련 서비스를 사용하게 된다면, 오로지 개발 툴인 이클립스를 활용해서 확인하는 수밖에 없다. 즉, 안드로이드 앱 개발 배포 후 유지보수를 위해 로그를 확인하려 해도, 개발자들은 매번 이클립스가 연결된 상태에서만 확인할 수 있는 것 이다.
따라서, 개발 전체 라이프사이클을 고려했을 때 꼭 필요한 backend service 들이 부족하다 보니, 실무에서는 많은 어려움을 겪고 있고 전체적인 품질 속성 및 개발 생산성과 연관이 되기도 한다.

1-3. 아키텍쳐 설계 개선 목적

본 모바일 오픈 아키텍처 개선 과제에서는 많은 개발자들과 오픈 소스 커뮤니티에서 검증된 오픈 소스들을 가져와서 개선/수정/리팩토링을 하여서 위에서 언급한 문제들 - 파편화, 폐쇄성, 사용성, 유지보수성 -을 해결하면서 다양한 요구사항을 만족시키고 개발 생산성 향상을 꾀하여서 개발 비용 관련 이슈나 파편화 문제 등을 해결하고자 한다.
더불어 이번 과제를 통해서 개선된 아키텍처 및 소스코드를 소개 및 공개/배포함으로써 다른 개발자들이 쉽게 활용하고 더 나은 feature, 아키텍처 및 생태계를 만들어 나갈 수 있도록 하였다.

1-4. 안드로이드 어플리케이션 아키텍처 (젤리빈 기준)



[그림 5. 안드로이드 어플리케이션 아키텍처]

안드로이드의 어플리케이션 계층의 아키텍쳐(4.1 기준)은 크게 3 가지의 레이어로 나뉜다. 커널 계층은 리눅스를 그대로 사용한 것이라 실제 모바일 개발과는 연관성이 낮으므로 분 문서에서 제외한다.

- Java 코어 계층
  Oracle (구 Sun) 사가 만든 공식 Java 라이브러리는 어플리케이션 최 하위 계층으로 사용하고 있다. 이것은 기존 Java 생태계를 잘 흡수하기 위한 포석으로 볼수 있다.

- Java 써드파티
  Java 표준은 아니지만, 앱 개발시 필요한 네트워크, XML 설정, 테스트등의 라이브러리를 탑재하였다.

- Andriod 플랫폼
  구글 사가 개발한 계층으로, 안드로이드 앱 개발시 필요한 범용적인 컴포넌트들을 제공한다. 각각의 컴포넌트들은 다음과 같다.

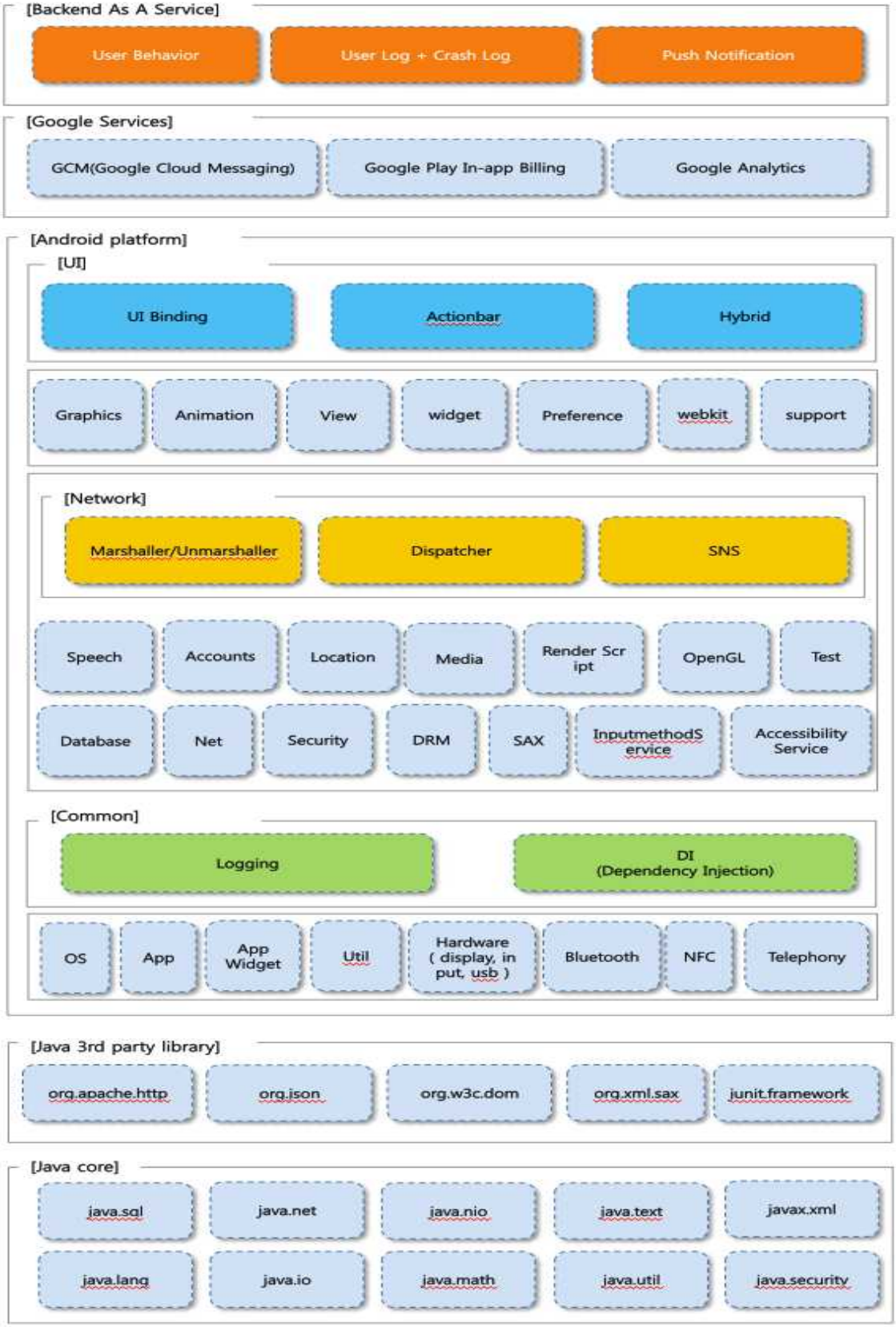
  - graphics - 그래픽 처리 중 저수준의 API 를 제공(canvases, color filters, points, rectangles 등)
  - animation - 애니메이션 처리를 쉽게할 수 있도록 기능 제공
  - view - 스크린에 표시될 레이아웃과 사용자와 상호소통할 수 있는 화면 제공
  - widget - 사용자 어플리케이션 내부에 사용될 UI 기본적인 구성을 제공하고 사용자가 변경가능하도록 제공
  - preference - 사용자 어플리케이션에서 설정을 쉽게할 수 있는 기능과 UI 제공
  - gesture - 제스쳐를 사용자가 정의할 수 있고 쉽게 사용할 수 있는 화면 제공
  - support - API 버전업이 되면서 UI 관련된 하위호환성을 위한 기능 제공
  - webkit - 웹브라우져 관련된 다양한 기능들 제공
  - speech - 말해서 입력하기 기능을 제공
  - accounts - 안드로이드 내부의 계정관리 기능 제공
  - location - 위치정보 관련된 기능 제공
  - media - 다양한 멀티미디어(audio, video) 관련된 기능 제공
  - renderscript - 3D 나 수학적 수식을 고성능으로 처리하기위한 저수준 API 제공
  - opengl - OpenGL ES 관련된 인터페이스 제공
  - test - 안드로이드 테스트 관련된 기능 제공

- database - 데이터베이스 관련된 기능 제공
- net - 기본적인 자바의 java.net.* API 대신 네트워크 관련된 기능을 쉽게 사용할 수 있도록 제공
- security - 안드로이드 보안 관련된 기능 제공
- drm - 보편적으로 사용할 수 있는 DRM 관련 기능 제공
- sax - SAX 핸들러 기능을 제공
- inputmethodservice - 입력기 관련된 API 제공
- accessibilityservice - 접근성을 높일 수 있는 장치 개발에 필요한 API 제공
- os - OS(operating system)의 기본적인 서비스들과 내부 통신을 처리함
- app - 안드로이드 어플리케이션 모델을 추상화하여 처리하고 관리함
- appwidget - 홈스크린에 표시되는 앱위젯을 관리함
- util - 보편적으로 사용하는 유틸리티성 메소드를 제공(date/time, base64 en/decoders, XML 등)
- hardware - 하드웨어 관련된 기능을 제공(camera, sensors 등)
- bluetooth - 블루투스 기능을 제공
- nfc - NFC(Near Field Communication) 기능을 제공
- telephony - 휴대전화 시스템의 기본적인 상태를 모니터링할 수 있도록 제공(통신사, 전화번호, 전화가능여부 등)

- 구글 서비스

  안드로이드 표준 라이브러리는 아니나, 앱 마켓을 이용하거나, 결재시 필요한 헬퍼 성격의 라이브러리들을 별도로 제공한다.
  - GCM(Google Cloud Messaging) - 사용자의 안드로이드 단말과 서비스 개발사가 구축한 서버간의 메세지를 할 수 있는 서비스 (푸시)
  - Google Play In-app Billing - 구글 플레이(마켓) 서비스이며 어플리케이션 내부에 있는 디지털 컨텐츠를 판매할 수 있는 시스템. 다양한 컨텐츠를 판매할 수 있다 (미디어파일, 사진, 가상 게임 포인트, 레벨, 프리미엄 기능들 등등)
  - Google Analytics - 어플리케이션 내부에서 만들어지는 데이터를 쉽게 수집할 수 있도록 만든 시스템이며, 웹어플리케이션에서 사용하는 구글 분석 서비스를 안드로이드에서도 사용할 수 있다.

1-5. 안드로이드 오픈소스 어플리케이션 블록 전체 아키텍처



[그림 6. 안드로이드 오픈소스 어플리케이션 블록]

본 문서에서 제공하는 모바일 어플케이션 모바일 블럭의 아키텍처는 4 가지의 계층으로 구성된다. 4 가지의 계층으로 선정한 이유는 UI, Network, Common, BaaS 가 모바일 어플리케이션 개발에 많은 비용 및 자원이 소모되기 때문이다.

- 유저 인터페이스 (User Interface)
  안드로이드의 화면 단편화및 버전 단편화로 인해 버전 별로 제공할수 있는 Widget 함수와 기능들이 상이하다. 액션바 같은 경우, 3.X 버전 때부터 안드로이드에 추가 되었지만, 2.X 버전의 사용자가 전체 안드로이드 폰에서 40%이상이므로, 실제 배포되는 앱에 적용할수 없다. 또한 안드로이드는 커스텀 유저 인터페이스 컴포넌트를 만드는데 많은 비용이 발생한다.

- 네트워크 (Network)
  독립형 어플리케이션보다 네트워크 기능을 제공하는 어플리케이션의 중요도가 (앱 내부 결제나 소셜 서비스의 연동)갈수록 성장하고 있다. 2012 년 Distimo 의 조사에 의하면 네트워크 기반의 어플리케이션의 매출이 전체 앱 시장에서 72%의 매출을 차지한다. 하지만 안드로이드가 제공하는 네트워크 기능은 단순해서, 어플리케이션 구축시 많은 비용이 발생하는 영역이다.

- 공용부 (Common)
  안드로이드는 구글의 폐쇄적인 정책으로 인해, 시장의 요구를 재빨리 반영하지 못하고 있다. 이로 인해 어플리케이션에 필요한 로깅, 결제 모듈, 컴포넌트 설정부등을 별도로 제공하지 않고 있다.

- 서비스로써의 백엔드 (BaaS)
  모바일 앱을 개발할때, 구글이 제공하는 백엔드 기능 (푸쉬, 사용자 통계, 오류 리포트)의 품질이 부족하다. 푸쉬 서비스 같은 경우는 지연이나 손실율이 발생하고, 에러 리포트 기능 또한 제한적인 정보만 제공한다. 그러므로 모바일 개발에 필요한 백엔드 서비스를 직접 구축해야 하는 문제가 발생한다.

본 문서는 위의 4 계층에서의 주요 문제의 해결책과 아키텍쳐적인 관점, 오픈소스 사례등을 설명하고 있다.

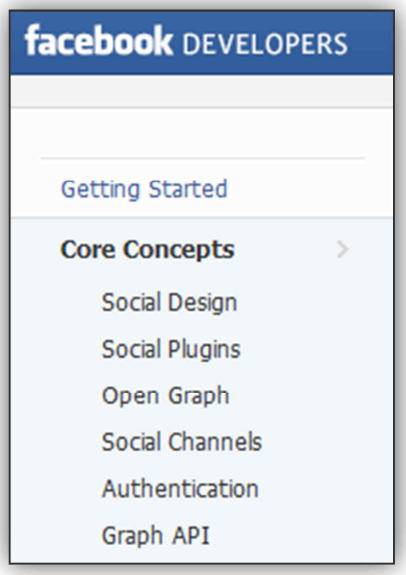
1. 모바일 오픈 아키텍처 레퍼런스 설계

2-1. SNS 연동

2-1-1. 개요 및 배경

현재 안드로이드 OS 를 비롯하여 iOS, 기타 웹에서 가장 많이 참조 및 사용하는 SNS 는 단연코 페이스북(facebook)이다. 페이스북은 10 억명의 사용하고, 한 사용자당 평균 30 분을 이용하는 충성도가 높은 서비스이다. 모바일 앱을 마켓팅하기 위한 최고의 플랫폼이다. 현재 모바일 개발 현장에서 페이스북을 활용하거나 직접 앱을 개발하려하지만, 사용성이 높은 페이스북 API 는 현재로서는 많이 존재하지 않는다.

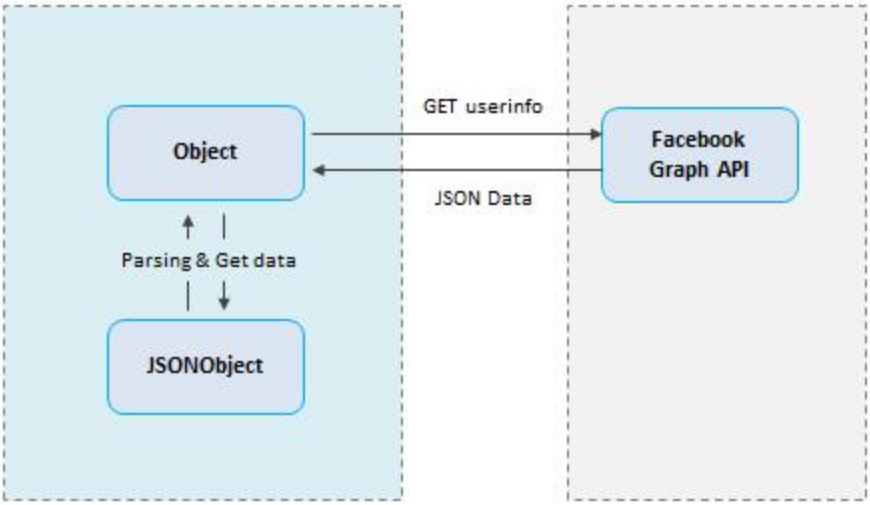
페이스북 앱을 만들기위해서는 5 가지의 구성요소(인증툴, 오픈 그래프, 소셜 채널, 소셜 플러그인, 그래프 API)를 숙지해야 한다. 이 구성요소를 잘 활용하기 위해서는 반복적이며, 많은 학습 노력이 선행되어야 한다.



[그림 7. 페이스북 활용을 위한 5 가지 구성 요소]

안드로이드 플랫폼을 활용한 모바일 SW 개발 시, 페이스북(facebook)과의 연동을 위해서는 json 데이터를 사용해야 하는데 파싱 과정에서 많은 개발 공수가 든다. 기존 facebook 에서 공식적으로 제공하는 안드로이드 SDK 를 이용할 경우, 프로토콜에 맞춰 복잡한 json 데이터를 생성해야 한다.

페이스북 연동을 위한 JSON 파싱은 아래 그림과 같은 구조로 이루어 지는데 자신의 정보를 얻어오는 그래프 API 를 요청한 다음 반환 값인 JSON 데이터 일일이 파싱하는 과정을 거친다



[그림 8. 페이스북 연동을 위한 JSON 파싱]

위의 구조처럼 페이스북의 데이터를 가져온다면 매번 JSON 파싱을 시도해야 하고 그 과정 또한 특별히 구조화 되어있지 않기 때문에 자연적으로 코드양도 증가할 수 밖에 없다. 그리고 복잡도도 증가하여 유지 보수과정에서도 문제가 발생하거나 공수가 증가할 가능성이 매우 높다.

2-1-2. 설계 개선 목표

SNS 연동에 대해서 사용성과 생산성 향상이라는 두 가지 목표를 가지고 설계를 개선하고 refactoring 하기로 했다.

개발자가 페이스북의 여러 컴포넌트를 잘 몰라도 쉽게 facebook 을 사용할 수 있게 만들었고, 단순히 객체의 메소드를 호출하는 것 만으로 쉽게 사용할 수 있는 클래스를 만들고자 하였다.
반복적인 json 파싱을 안하기 때문에 빠르고 편리하게 개발할 수 있고, 초급 개발자가 활용하더라도 쉽게 개발 할 수 있도록 직관적으로 refactoring 하고자 했다.

본 문서에서는 페이스북의 모든 프로토콜을 다 추상화할수 없어서 사용자가 특별히 많이 사용하는 부분 몇 개를 선별하여 추상화 작업을 시작하였다.

2-1-3. 아키텍쳐 개선을 위한 사전 활동

2-1-3-1. 아키텍쳐 개선의 시작

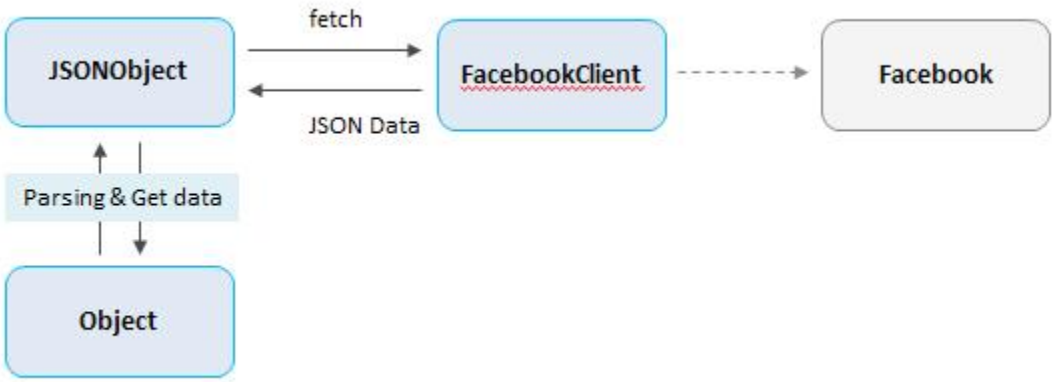
기존의 이키텍처를 개선하기 위한 활동에는 아래와 같이 여러가지가 방법이 있을 수 있다.

- 새롭게 아키텍처를 설계하고 구성한다
- 기존에 있는 아키텍쳐를 refactoring 하여 사용성이 높은 아키텍처를 설계한다
- 기존 문제점들을 인식하여 개선하고자 노력한 오픈 소스를 적극적으로 활용한다

근래에 폭발적으로 성장하고 있는 오픈 소스 생태계에는 다양하고 품질도 우수하면서 사용하기 좋은 오픈 소스들이 많이 늘어나고 있으며, 또한 직접 개발하고 기여하면서 오픈 소스 생태계에 참여하여 더 좋은 소프트웨어를 만들어 낼 수 있다.
게다가 전 세계의 다양한 개발자들이 참여하여 개발하므로 개인 혼자 설계하는 아키텍쳐보다 월등한 양질의 설계 품질을 기대할 수 있고 다양한 채널과 커뮤니케이션을 통해서 개선을 위한 협업을 추진할 수 있다.
따라서, 본 장에서는 SNS 연동을 위한 아키텍쳐 개선 작업을 위하여 오픈 소스를 활용, refatoring 하여서 기존 문제점을 해결하고 사용성이 높은 아키텍쳐를 구현하고자 한다.

2-1-3-2. 활용할 오픈 소스 선택

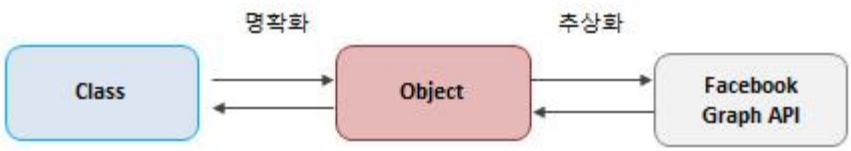
먼저 기존의 오픈 소스를 잘 활용하는 측면에서 기존에 나와 있는 몇몇 자바 기반의 페이스북 프레임워크의 아키텍쳐를 분석하였다.
첫번째로 페이스북 SDK 를 조사하였으나, 그 구조를 보았을 때 패키지화 하지 않았고 클래스로만 코드를 작성하였다. 또한, JSON 통신 이여서 편하게 가져다가 사용하기 힘든 문제가 있었다. 위의 세가지 문제로 인해 사용할 오픈 소스 대상에서는 제외하기로 했다.
두번째로 자바 페이스북 프레임워크인 restFB 를 조사하였다. restFB 도 사용하기는 힘들었다. 기본적인 패칭, 즉, 어떤 메소드에 정보를 달라고 하면 facebook 시스템을 완벽히 이해해야만 사용할 수 있는 문제가 있었다. URL 단위로 완전히 알고 있어야 했고, json 도 일일히 다 파싱을 해야 했다. 아래 샘플 코드를 보면 restFB 를 사용하려면 많은 양의 코드 작업과 시간 투입, 그리고 json 사용이 필요하다는 것을 알 수 있다.



[그림 9. restFB 의 JSON 통신 아키텍쳐]

하지만, 안드로이드 OS 에 익숙한 개발자들을 위한 자바로 개발된 facebook 연동 오픈 소스 라이브러리는 restFB 가 가장 대표적이었고, 활용성의 문제를 제외하고는 성능에 강점을 보였으며 이미 많은 사람들이 사용하고 있어서 검증이 되었다는 장점이 있어서, 본 문서에서는 restFB 오픈 소스를 fork 한 후 리팩토링 해서 더 좋게 개선하고자 하였다.

2-1-3-3. 사용성 개선 방향 도출



[그림 10. 사용성 개선 방향성 도출]

사용성 개선은 개발자가 쉽게 API 를 사용하는 것을 최우선 목표로 해서 기본 개념을 설정하고, 페이스북에서 사용하는 중요 기능을 선정해 샘플 코드를 작성하면서 시나리오를 도출해 나갔다.

1. 생성자 오버로딩

```
User user;
User user = new User(); // 자기자신
User user = new User("ID_STRING"); // 타 유저
```

2. 사용자 ID(user, page...)와 반환객체 맵핑

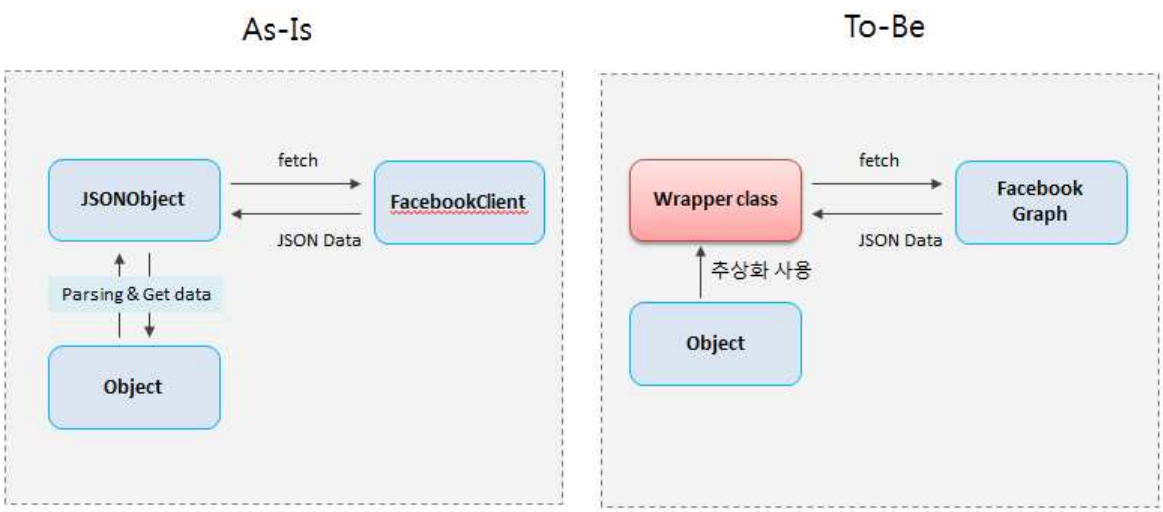
```
restfb방식
        User user = facebookclient.fecthObject("me",User.Class);
        User user = facebookclient.fecthObject("ID_STRING",User.Class);
새로운방식
        User user=getInfo.user(); //자기자신
        User user=getInfo.user("ID_STRING");
        User user=getInfoForUser();
        Page page =getInfoForPage();


    //Group관련 매니저 객체 생성, 매니저 객체를 통해서 group 관련 api 사용가능
    GroupManager groupMgr = new GroupManager(accessToken);

    //User관련 매니저 객체생성, 매니저 객체를 통해서 User 관련 api 사용가능
    UserManager userMgr = new UserManager(accessToken);

    // gruop 정보 가져오기
    Group group = groupMgr.getInfo(groupId);

    //사용자 정보 가져오기
    User user = userMgr.getInfo(userId);
    //user = userMgr.getMyInfo();          //자신의 정보를 가져올 경우

    //타입에 상관없이 (Photo, Checkin, User .. ) 정보를 가져올때는 위처럼 getInfo를 이용한
```

[그림 11. 사용성 개선 방향]

2-1-4. 아키텍쳐 개선 활동

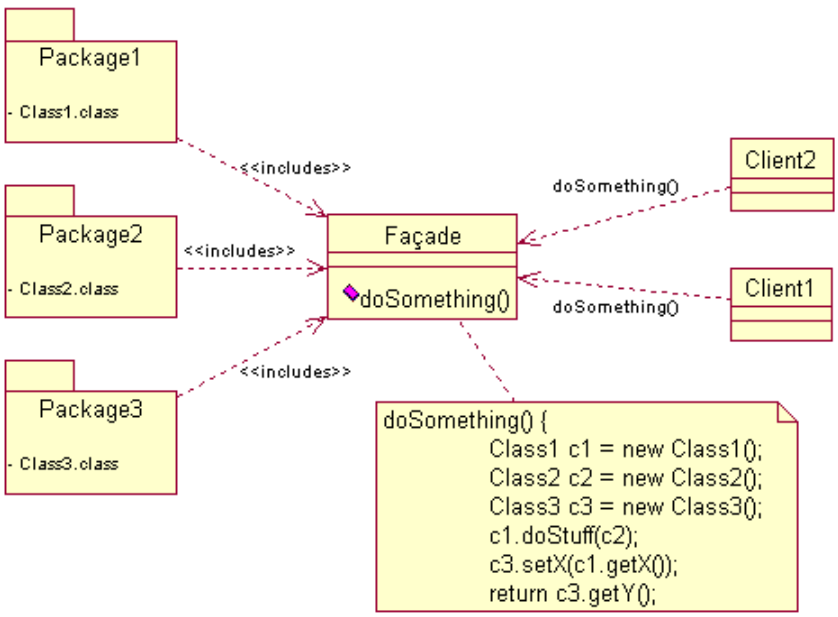
2-1-4-1. 구조 개선

보다 구조적이고 solid 한 아키텍쳐를 만들기 위해서, 기존 구조를 분석해서 문제점을 파악하여 간단한 구조부터 만들고자 했다.



[그림 12. 안드로이드 오픈소스 어플리케이션 블록]

본 문서에서는 래퍼 퍼사드 패턴을 이용하여 restFB 를 리팩토링하였다. 래퍼 퍼사드 패턴은 클라이언트들이 좀 더 쉽게 사용하고 접근할 수 있도록 해주는데 목적이 있는 패턴이고, 그 대표적인 예가 winAPI 의 클래스를 사용하기 쉽도록 하기 위해 wrraping 하여 MFC 클래스라고 할 수 있다.



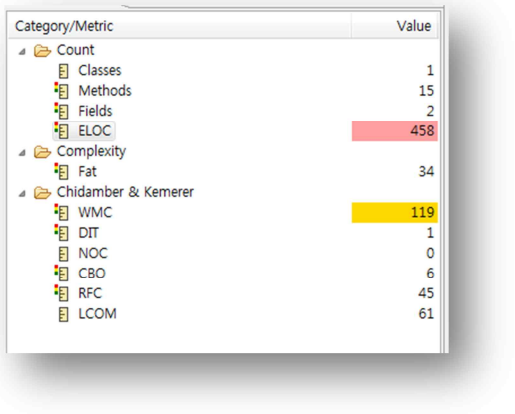
[그림 13. wrapper facede 패턴]

사용성을 강조하기 위해서 일반 사용자가 어플리케이션의 샘플만 이용하더라도 함수를 호출하여 사용할 수 있도록 만들었다.

기존 레거시 시스템의 자유도는 보장하면서, 시스템 사용자에게는 높은 개발 생산성을 보장하기 위해, Wrapper Façade 패턴을 적용하여 이 문제를 해결했다.

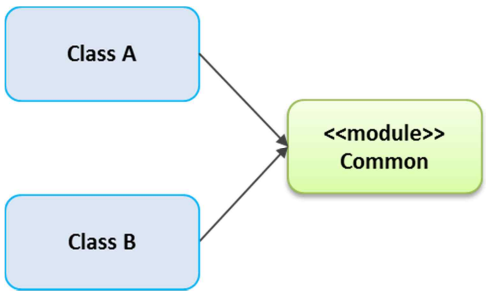
2-1-4-2. 복잡도 개선

아래 그림에서 보여지는 restFB 의 DefalutJsonMapper 라는 클래스는 기본적인 구조가 복잡한 것을 확인할 수 있고, STAN 이라는 툴을 이용하여 문제를 수치화하여 보면 ELOC (Estimated Line of Code) 코드의 양이 매우 많고, WMC(Weighted Methods per Class)클래스의 함수의 무게가 무겁다는 문제를 확인할 수 있다.

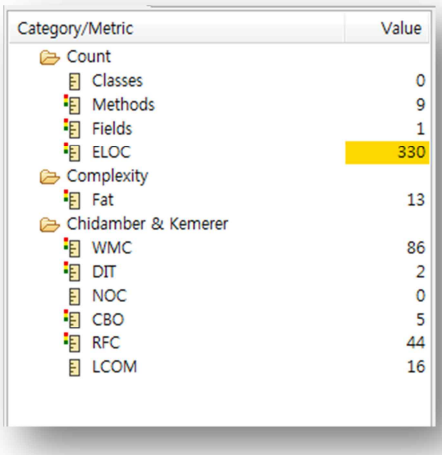


[그림 14. STAN 으로 분석한 restFB 의 복잡도]

특정 클래스가 지니차게 많은 기능을 수행하여, 많은 메소드나, 긴 메소드를 가지는 문제를 해결하기 위해 클래스를 책임별로 분리하고, 공통 기능 부분은 공용부로 추출하는 방법을 통하여 리팩토링을 진행하여 현저히 수치가 향상되었다.



[그림 15. 공통 부분 모듈화]



[그림 16. 리텍토링을 통해 복잡도 개선]

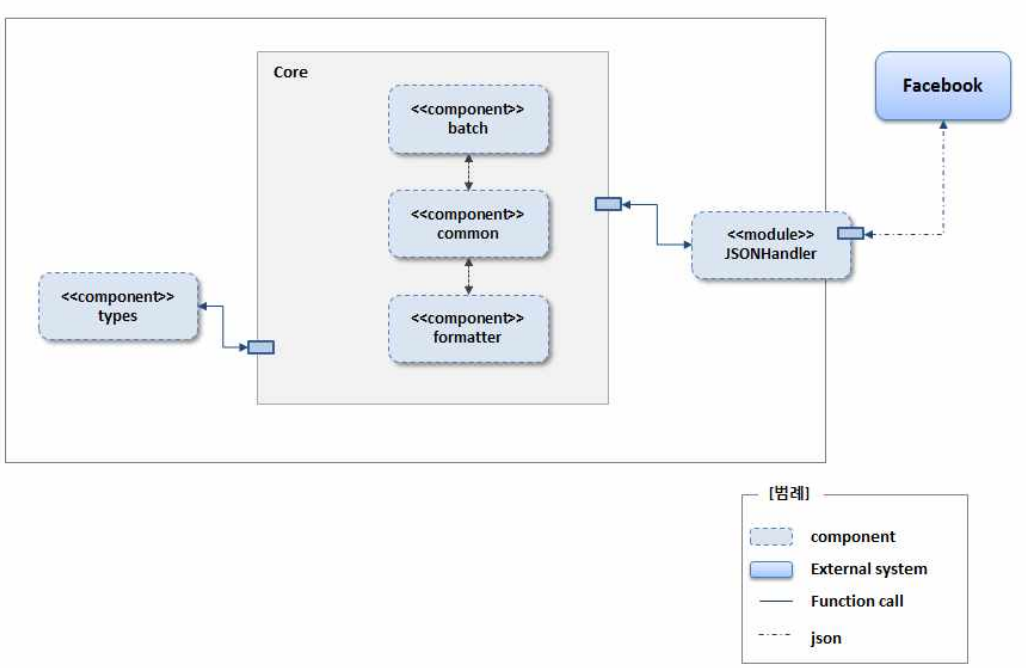
2-1-4-3. 가독성 있는 네이밍 정의

기존의 프레임워크 단에서 친구 데이터를 불러오기 위해서는 먼저 restFB 의 함수를 호출하는 방식과 그리고 페이스북 api 에서 호출해야 하는 connection 타입을 모두 알아봐야 하는 번거로움이 있었고, 또한 리턴되는 데이터가 기존 프레임워크에 정의되어 잇지 않다면 json 데이터의 태그를 읽어들어 매핑해줘야 했다.
따라서, 가독성있는 네이밍과 함수 호출 방식에 신경을 썼고, 나의 친구들의 목록을 불러올 때는 User 객체에 friends 란 함수를 호출하여 불러 올 수 있도록 구현하였다. 또한 페이스북에서 제공하는 오브젝트들을 모두 매핑시켰다.

```
user = new User();
user = user.createInstance("me");

Connection<Friends> friends = user.friends();

for (List<Friends> myFriends : friends)
    for (Friends users : myFriends)
        System.out.println("friends: " + users.getName());
```

[그림 17. Naming 정의 개선]

2-1-5. 아키텍쳐 개선 결과



[그림 18. Facebook 프레임워크 개선 셜과]

기존 Facebook 시스템과의 연동을 위해 사용자 코드에 JSON 데이터를 요청하고, 파싱하는 코드들이 삽입되어, 가독성과 생산성이 저해되었다.
이러한 문제를 해결하기 위해, 기존 오픈소스 restFB 를 캡슐화하여 다음과 같이 컴포넌트들을 구성하였다.

- JSON Handler: 페이스북 시스템과 JSON 데이터 통신을 전담한다. 또한 개발자는 JSON 데이터를 알 필요가 없다.
- Core: 개발자가 쉽게 객체로 핸들링 할수 있게 하기 위해서 JSON 데이터를 분석한 후 객체로 변환하고, 반대로 객체로 전달한 데이터를 JSON 형태로 변환하는 역할을 한다.
- Types: Post, Like 와 같이 페이스북의 언어로 객체를 맵핑하여 제공한다.

2-1-5-1. 개발 생산성 개선

“친구 리스트 가져오기“나 “피드 올리기“ 기능으로 봤을 때 개발 생산성 개선이 이루어 졌다.
- 기존 페이스북 API

```
HttpsURLConnection connection = null;
BufferedReader bufferedReader = null;

URL url = new URL("https://graph.facebook.com/me/friends?access_token=ACCESS_TOKEN");
connection = (HttpsURLConnection) url.openConnection();

connection.setRequestMethod("GET");
connection.setRequestProperty("Content-Type" , "application/x-www-form-urlencoded");
connection.connect();

InputStreamReader reader = new InputStreamReader(connection.getInputStream());
bufferedReader = new BufferedReader(reader);

String temp = null;
StringBuffer buffer = new StringBuffer();
while ((temp = bufferedReader.readLine()) != null) {
    buffer.append(temp);
}
JsonObject json = new JsonObject(buffer.toString());
JsonArray jsonArray = json.getJsonArray("data");
for(int i=0;i<jsonArray.length();i++){
    JsonObject jsonObject = jsonArray.getJsonObject(i);
    System.out.println("My friends : " + jsonObject.get("name"));
}

connection.disconnect();
```

- 개선한 API

- fHalo
  - Connection<Friends> friends = user.friends();

- 기존 페이스북 API

```
BufferedReader bufferedReader = null;
HttpURLConnection connection = null;

URL url = new URL("https://graph.facebook.com/me/feed?access_token=ACCESS_TOKEN")
connection = (HttpURLConnection) url.openConnection();

connection.setDoOutput(true);
connection.setRequestMethod("POST");

String parameter = "";
parameter = URLEncoder.encode("message", "UTF-8") + "=" + URLEncoder.encode("Message Test", "UTF-8
parameter += "&" + URLEncoder.encode("description", "UTF-8") + "=" +  URLEncoder.encode("Descripti
parameter += "&" + URLEncoder.encode("caption", "UTF-8") + "=" + URLEncoder.encode("Caption Test",

OutputStreamWriter outputStreamWriter = new OutputStreamWriter(connection.getOutputStream());
outputStreamWriter.write(parameter);
outputStreamWriter.flush();

bufferedReader = new BufferedReader(new InputStreamReader(connection.getInputStream(), "UTF-8"));
String line = null;
StringBuffer buffer = new StringBuffer();
while((line = bufferedReader.readLine()) != null) {
    System.out.println(line);
    buffer.append(line);
}

JSONObject json = new JSONObject(buffer.toString());
System.out.println("id : " + json.get("id"));
```

- 개선된 API

- fHalo
  - Feed feed = new feed();
    feed.setMessage("Message Test"); feed.setCaption("Caption Test"); feed.setDescription("Description Test");
    user.publishFeed(me, feed);

2-2. Simple Framework, Gson

2-2-1. 개요 및 배경

현재 인터넷에서 가장 일반적으로 사용되고 있는 데이터 형식으로 XML 과 JSON(JavaScript Object Notation)이 있다. 두 가지 데이터 형식은 모바일과 서버 또는 모바일과 웹사이에서 자료를 주고받을 때 그 자료를 표현하는 방법이며, 안드로이드 플랫폼에서는 기본적으로 XML 과 JSON 형식의 데이터를 파싱하기 위한 자바 라이브러리를 제공하고 있다.

안드로이드 플랫폼을 활용한 모바일 소프트웨어 개발 시, XML 이나 JSON 형식을 통해 서버와 데이터를 주고 받게 된다. XML 과 JSON 형식의 데이터는 모바일 소프트웨어가 직접적으로 사용할 수 없기 때문에 필요한 데이터로 변환하는 파싱과정을 거쳐야 하는데, 이 과정에서 많은 개발 공수가 든다.

XML 과 JSON 형식의 데이터 파싱은 태그 또는 키 값을 읽어들어 오브젝트에 직접 매핑해줘야 하므로, 데이터 형식이 복잡해 질수록 코드의 복잡도는 증가하게 된다. 또한 프로토콜이 변화하게 되면, 파싱하는 내부 코드도 변경해줘야 하기 때문에 유지 보수 과정에서도 문제가 발생하거나 공수가 증가할 가능성이 매우 높다.

2-2-2. 설계 개선 목표

XML 과 JSON 형식의 데이터를 파싱하는데 발생하는 기존 문제를 해결하자고 생산성 향상과 변경 용이성이라는 두 가지 목표를 가지고 오픈 소스를 선정하기로 했다.

2-2-3. 아키텍쳐 개선을 위한 사전 활동

2-2-3-1. 아키텍쳐 개선의 시작

본 장에서는 심플 프레임워크와 GSON 프레임워크를 활용하여 기존의 XML 및 JSON 형식의 데이터를 파싱하는 문제점을 해결하여 생산성과 프로토콜 변화에도 유연하게 대체할 수 있는 용이성 측면에 집중하였다.

2-2-3-2. 오픈 소스 선정 이유

안드로이드 플랫폼에서는 XML 과 JSON 형식의 데이터 파싱하기 위해 XML 과 JSON 형식에 대한 파서 함수를 제공한다.

각각의 데이터 파서는 XML Pull Parser 방식과 JSON Parser 이며, 이 방식을 통해 Tag 나 Name 값을 읽어주고, 객체의 파라메터와 데이터를 매핑해주는 과정을 거치게 된다. 즉 XML 이나 JSON 에 추가되는 정보가 많아지거나 파싱해야 하는 데이터의 구조가 증가하면 그만큼 코드의 양도 비례하여 증가하게 되며, 본 문서에서 설계한 생산성 향상과 변화 용이성면에서 어긋나는 구조가 된다.

하지만, Simple Framework 와 GSON Framework 는 XML 과 JSON 형식의 데이터를 파싱하는데 있어서 기존의 문제점을 해결할 수 있는 프레임워크였다. 기존 파싱 방법과 달리 객체 클래스를 통해 데이터를 변환하기 때문에 기존 코드에 비해 매우 간단한 구조를 가지고 있으며, 프로토콜이 변화하더라도 객체 클래스만 변경해주면 되기 때문에 변화 용이성 측면에서 뛰어난 장점을 지닌다.

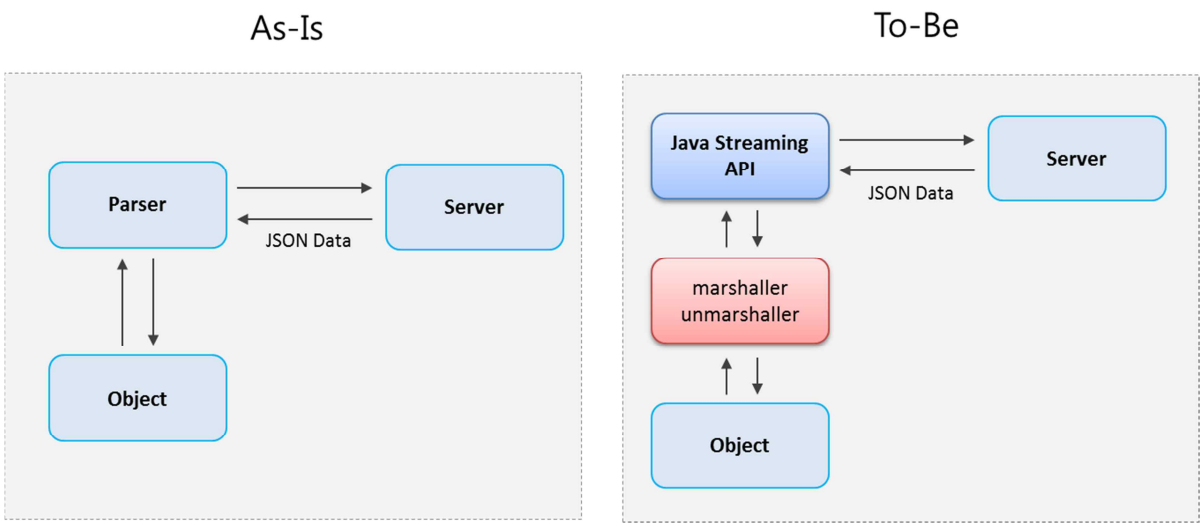
2-2-4. 아키텍쳐 개선 활동

2-2-4-1. 구조 개선

1) composite message 패턴

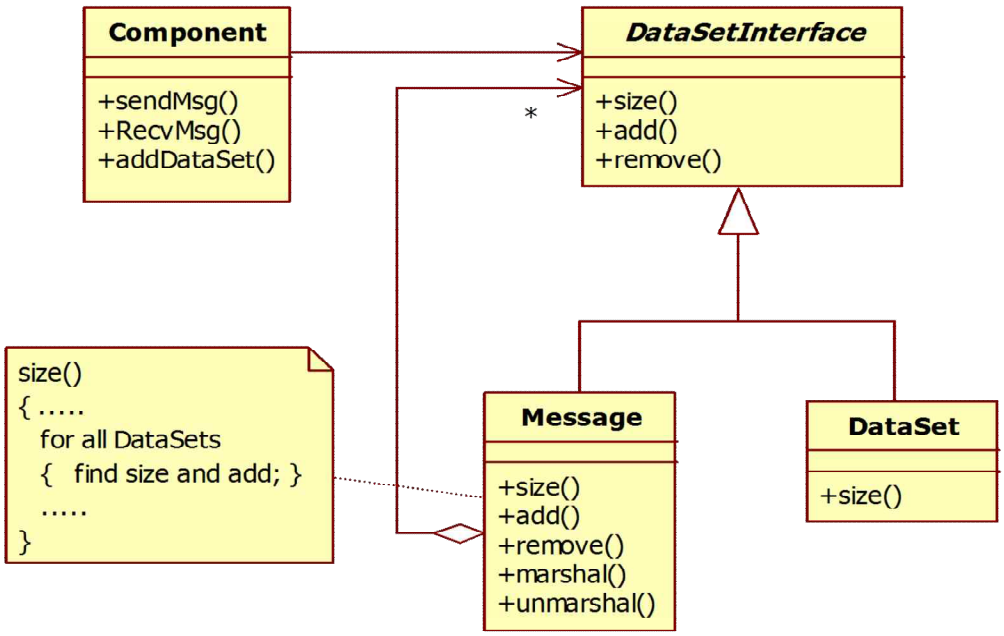
기존 모바일 SW 에서 제공하는 라이브러리의 구조와 본 문서에서 선정한 자바 프레임워크의 구조를 비교하였다.

기존 구조는 JSON (또는 XML) 형식의 데이터를 스트링으로 변환한 후 파서를 통해 객체 형태로 변환되는 구조를 가지는 반면, 심플 프레임워크와 Gson 프레임워크는 마샬러를 통해 데이터가 객체로 변환되는 구조를 가진다.



[그림 19. 안드로이드 오픈소스 어플리케이션 블록]

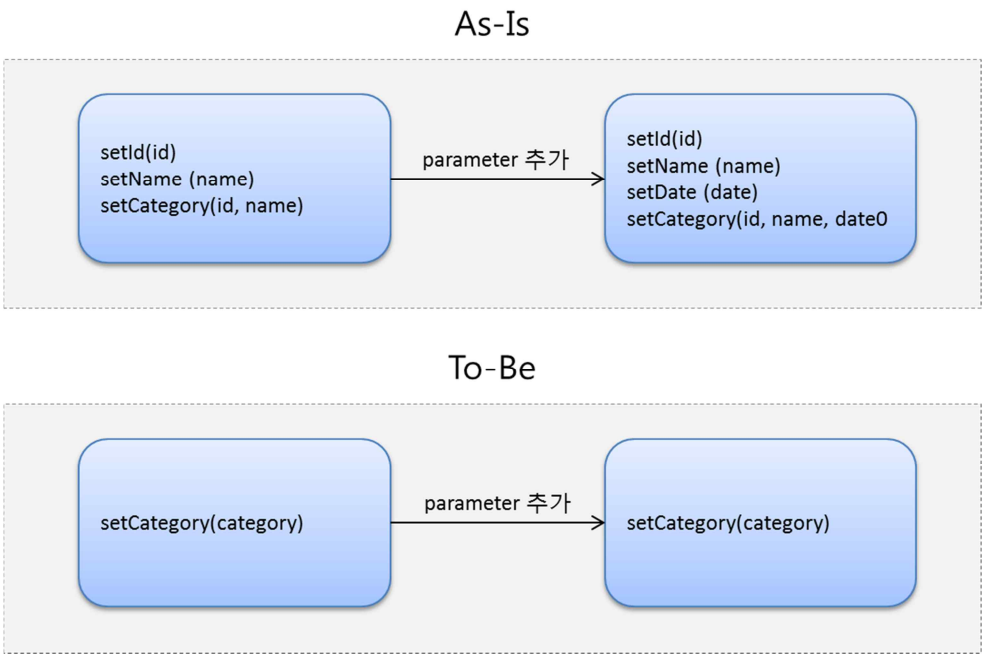
즉, 본 문서를 통해 기존 구조를 개선한 프레임워크는 컴포지트 메시지 패턴을 이용하였음을 확인할 수 있다. 컴포지트 메시지 패턴은 새로운 프로토콜 추가되더라도 유연하게 대처할 수 있는 패턴으로, 간단한 코드를 가지고 반복적인 작업을 진행할 수 있으므로 작업의 생산성을 높일 수 있다.



[그림 20. composite message 패턴]

2) Parameter Object

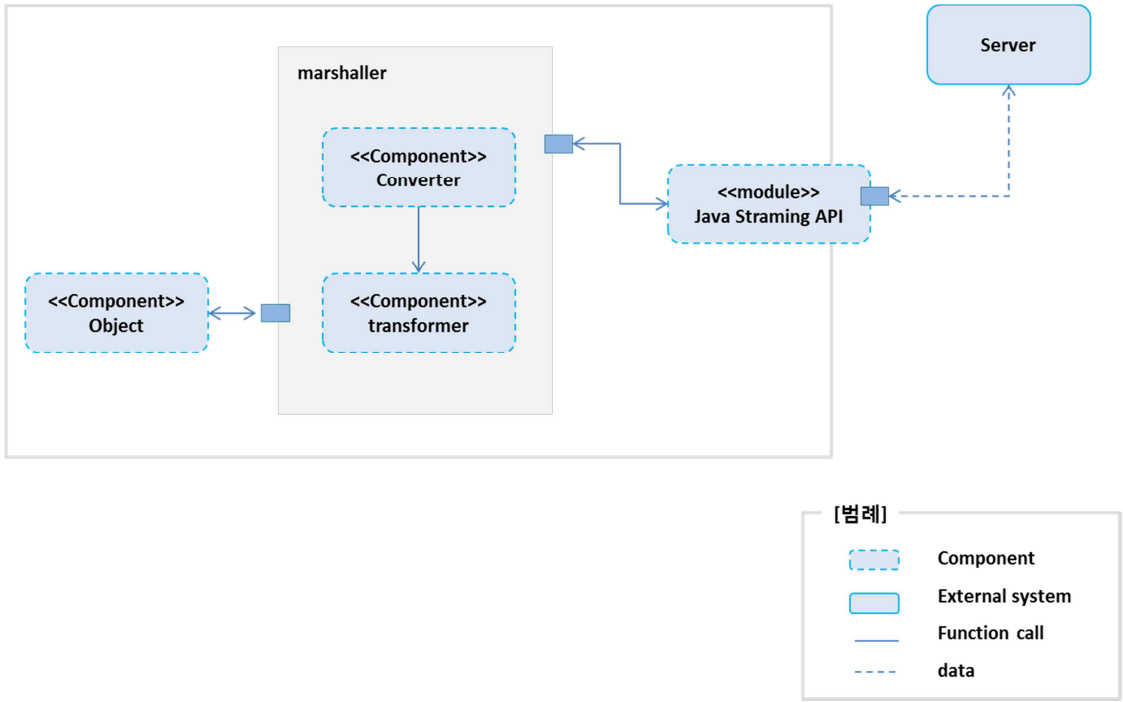
기존 구조에서 JSON 또는 XML 데이터 파서는 프로토콜이 변경되면 데이터를 매핑해 줄 수 있는 코드를 추가해줘야 하므로 유지 보수 과정에서 많은 공수가 필요하다.



[그림 21. 안드로이드 오픈소스 어플리케이션 블록]

하지만 심플 프레임워크 또는 Gson 프레임워크는 데이터의 구조나 프로토콜이 변하더라도 내부 코드의 변화는 없다. XML 이나 JSON 데이터의 구조가 어찌됐든 또 어떻게 변화하든 내부 코드는 하나의 객체를 넘겨주고 리턴받는 구조이기 때문이다.
즉 파라미터 오브젝트를 사용해 객체의 데이터를 잘 취합함으로써 새로운 변화를 잘 흡수할 수 있는 유연한 구조를 가지므로, 변경 용이성 측면에서 장점을 갖는다.

2-2-5. 아키텍쳐 개선 결과



[그림 22. 안드로이드 오픈소스 어플리케이션 블록]

기존 파서 시스템의 구조를 개선하기 위한 프레임워크의 구조는 다음과 같은 컴포넌트들로 구성되어 있다.

- Converter: JSON 또는 XML 형식의 데이터를 직렬화 해주는 역할을 한다.
- Transformer: 직렬화된 데이터를 분석하고, 파라미터와 데이터를 매핑해주어 객체로 변환해 주는 역할을 한다.
- Object: 개발자가 변환하고자 하는 객체로 결과를 반환한다.

2-2-5-1. 개발 생산성 개선

1) XML

다음은 XML Pull Parser 방식을 통하여 XML 형식의 데이터를 객체에 매핑해 주는 예제 코드이다.

```
int eventType = parser.getEventType();
  String tag = "" ;
  while (eventType != XmlPullParser.END_DOCUMENT) {
        switch (eventType) {
        case XmlPullParser.START_TAG:
```

```
                    tag = parser.getName();
                    if(tag.equals("category")){
                        category = new Category();
                    }
                    break;

                case XmlPullParser.TEXT:

                    if(tag.equals("id")){
                        category.setId(Integer.parseInt(parser.getText()));
                    }else if(tag.equals("name")){
                        category.setName(parser.getText());
                    }else if(tag.equals("date")){
                        category.setDate(parser.getText());
                    }
                    break;

                case XmlPullParser.END_TAG:
                    String endTag = parser.getName();
                    if(endTag.equals("category")){
                        categoryList.categories.add(category);
                    }
                    break;

                default:
                    break;
                }
                eventType = parser.next();
        }
```

아래는 심플 프레임워크를 통해 XML 형식의 데이터를 객체로 변환해 주는 예제 코드이다.

```
Serializer serializer = new Persister();
Reader reader = new StringReader(xml);
CategoryList categoryList = serializer.read(CategoryList.class, reader, false);
```

위에서 작성한 코드에서 볼 수 있듯이 Simple Framework 를 통한 파싱 방법은 XML Pull Parser 방식에 비해 매우 간단하다는 것을 확인할 수 있다.
또한 XML Pull Parser 방식은 XML 형식의 구조가 복잡해 질수록 if~else 문을 통해 태그 값과 객체의 파라메터를 매핑시켜줘야 하는 반면, 심플 프레임워크를 통한 파싱 방식은 XML 의 구조가 복잡해져도 내부의 코드는 바꿀 필요가 없다.

2) JSON
다음은 Json Parser 방식을 통하여 Json 형식의 데이터를 객체에 매핑해 주는 예제 코드이다.

```
JSONObject jsonObject = new JSONObject(json);
JSONArray array = jsonObject.getJSONArray("categories");

for(int index =0 ; index < array.length() ; index++ ){
        category = new Category();

        JSONObject categoryObject = array.getJSONObject(index);

        category.setId(categoryObject.getInt("id"));
        category.setName(categoryObject.getString("name"));
        category.setDate(categoryObject.getString("date"));

        categoryList.categories.add(category);

}
```

아래는 GSON 프레임워크를 통해 JSON 형식의 데이터를 객체로 변환해 주는 예제 코드이다.

```
Gson gson = new Gson();

CategoryList categoryList = gson.fromJson(json, CategoryList.class);
```

XML 파서 방식과 마찬가지로 Gson 프레임워크를 통한 Json 데이터 파싱 방법은 안드로이드 플랫폼에서 제공하는 JSON 파서 방식에 비해 매우 간단하다.
JSON 파서는 JSON 형식의 데이터를 배열 또는 객체 형식으로 변환한 후, 키값을 읽어들여 각각 객체의 필드에 매핑시켜줘야 한다.

하지만 Gson 프레임워크를 통한 파싱 방식은 함수를 통해 원하는 객체로의 변환이 가능하다. 즉, JSON 의 데이터 구조가 복잡해져도 내부의 코드는 바뀔 필요가 없으므로 코드 작업의 생산성을 높혀준다.

2-3. IoC Framework, RoboGuice, AndroidAnotations

2-3-1. 개요 및 배경

안드로이드의 장점은 오픈 소스로 되어있어서 많은 단말 제조사에서 안드로이드를 자신의 제품에 입맛에 맞게 적용할 수 있다는 점이다. 피쳐폰이 중심이던 시절에 아이폰이 출시되었고 그 대항마로 안드로이드를 선점했던 이유는 다 이러한 이유가 있기때문이다. 그래서인지 Samsung, LG 등 많은 OEM 제조사들이 안드로이드를 채택하고 현재 2012 년 9 월 통계에 따르면 안드로이드 기기들이 50.8%로 분포하고 있는 것을 알 수 있다.

Top Smartphone Platforms
3 Month Avg. Ending Jul. 2012 vs. 3 Month Avg. Ending Apr. 2012
Total U.S. Smartphone Subscribers Ages 13+
Source: comScore MobiLens

	Share (%) of Smartphone Subscribers		
	Apr-12	Jul-12	Point Change
Total Smartphone Subscribers	*100.0%*	*100.0%*	*N/A*
Google	50.8%	52.2%	1.4
Apple	31.4%	33.4%	2.0
RIM	11.6%	9.5%	-2.1
Microsoft	4.0%	3.6%	-0.4
Symbian	1.3%	0.8%	-0.5

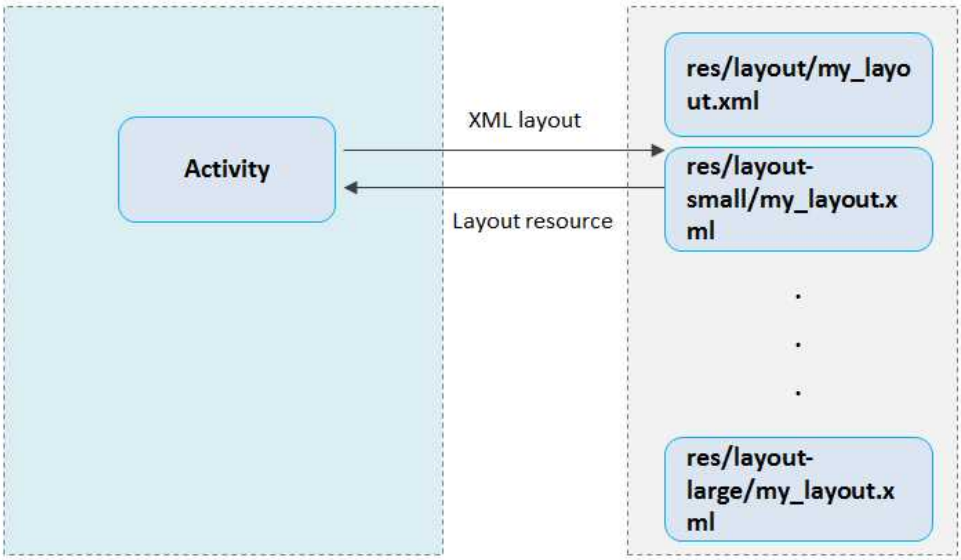
[그림 23. comScore 통계 - http://mashable.com/2012/09/05/android-52-percent]

하지만, 안드로이드 장점이 안드로이드 어플리케이션을 개발하는 개발자에게는 단점으로 다가온다. 왜냐하면 다양한 해상도, 다양한 기기의 특성을 고민해야 하기 때문이다. 애플의 iOS 제품들은 몇 개 되지 않는 것과 대조적이다.

오픈시그널맵스가 발표한 자료에 따르면 무려 3997 개의 저마다 다른 안드로이드 기기가 출시되었고, 화면크기부터 모바일 프로세서, 카메라, 센서, 운영체제가 모두 다른 제품들이 출시되었고 사용자가 사용중이다.

그래서인지, A 라는 기기에서는 동작하지만, B 라는 기기에서는 동작하지 않거나 UI 화면이 맞지 않아서 사용자경험(UX)를 충분히 줄 수 없어 제품의 매력이 떨어지게 된다. 안드로이드 플랫폼 내부적으로 이러한 문제점을 인지하고 있고, 그 대안들을 찾아내기 시작하였다.

다양한 해상도가 존재하기 때문에, 단말 기기 스크린의 밀도와 상관없이 상대적인 거리와 위치를 표현할 때 필요한 개념인 DIP 를 만들어 내었고 모든 안드로이드 개발자 혹은 디자이너는 DIP 에 기반한 기획과 개발을 진행을 해야 한다.
또한, 다양한 기기를 지원하기 위해서 레이아웃(layout) XML 파일들과 아이콘(icon)들이 UI 관련된 내용을 개발자가 직접 구현해야 한다는 것이다. 이러한 UI 뿐아니라 안드로이드 어플리케이션을 개발하면 반복되는 구현들을 너무나 많이 접하게 된다.



[그림 24. 안드로이드에서의 레이아웃 처리방법]

다양한 해상도를 대응하기 위한 안드로이드의 플랫폼의 노력에도 변하지 않는 것들 것 있다. 바로, 객체간의 의존관계가 밀접하게 있어 유지보수하기 어려워진다는 점과 안드로이드 플랫폼에 의존적인 코드가 생성되고 그 코드가 반복적으로 구현된다는 점이다. 반복적인 구현은 복잡도 증가로 이어져 유지 보수과정에서도 문제가 발생하거나 공수가 증가할 가능성이 매우 높다.

2-3-2. 설계 개선 목표

안드로이드에서 반복적인 코드를 줄임으로써 복잡도를 낮추고 생산성을 향상하는 두 가지 목표를 가지고 진행하기로 했다.
그에 따라, 본 문서에서는 대표적인 안드로이드의 반복적인 작업을 선정하고 이를 개선하는 것을 목표로 하였다.
가) UI 매핑

나) 파라미터 처리
다) 비동기처리 (Async)
리) REST 통신 (Http)

개발 복잡도를 낮추는 방법은 기존의 과거에도 많이 사용한 IoC(Inversion of Control) 개념을 적용하여 객체 서로간의 결합도를 낮추게 될 것이다.

2-3-3. 아키텍쳐 개선을 위한 사전 활동

2-3-3-1. 아키텍쳐 개선의 시작

본 장에서는 반복적인 작업을 줄여주고 객체간의 의존성을 낮추어줄 수 있는 아키텍쳐 개선 작업을 위하여 오픈 소스를 활용하여 기존 문제점을 해결하고 복잡도가 낮고 유지보수성이 높은 아키텍쳐를 구현하고자 하였다.

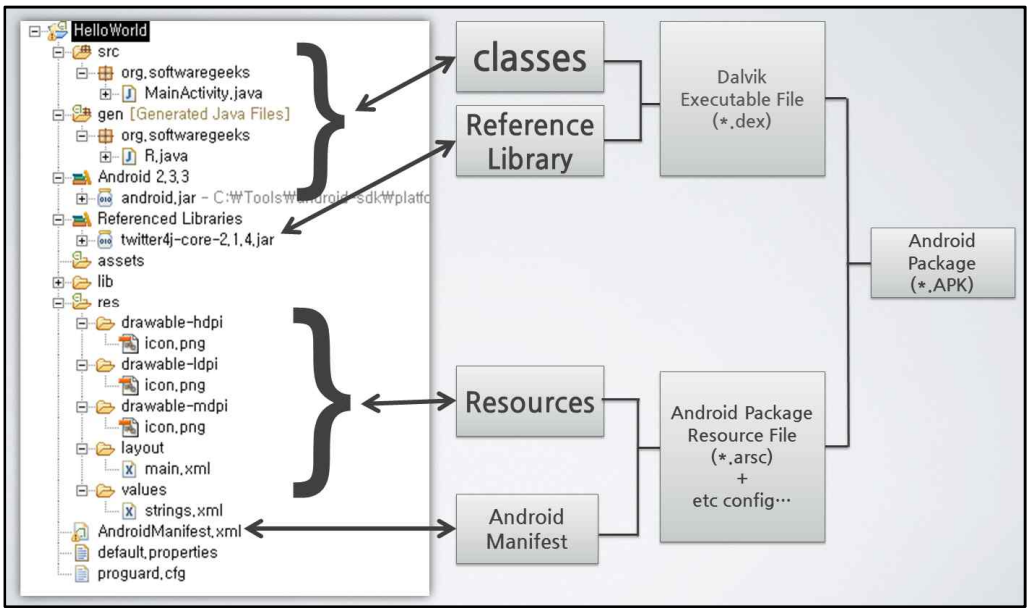
2-3-3-2. 기술 근간 조사

1) 코드 중복의 원인 분석

반복적이고 중복되는 코드가 무엇 때문에 발생하는지 알아야지 개선을 할 수 있기때문에 먼저 조사를 해보았다. 중복코드의 문제점은 안드로이드 플랫폼의 구조와 Java 언어적인 문제점 크게 2 가지로 조사되었고 각 항목들을 정리하면 다음과 같다.

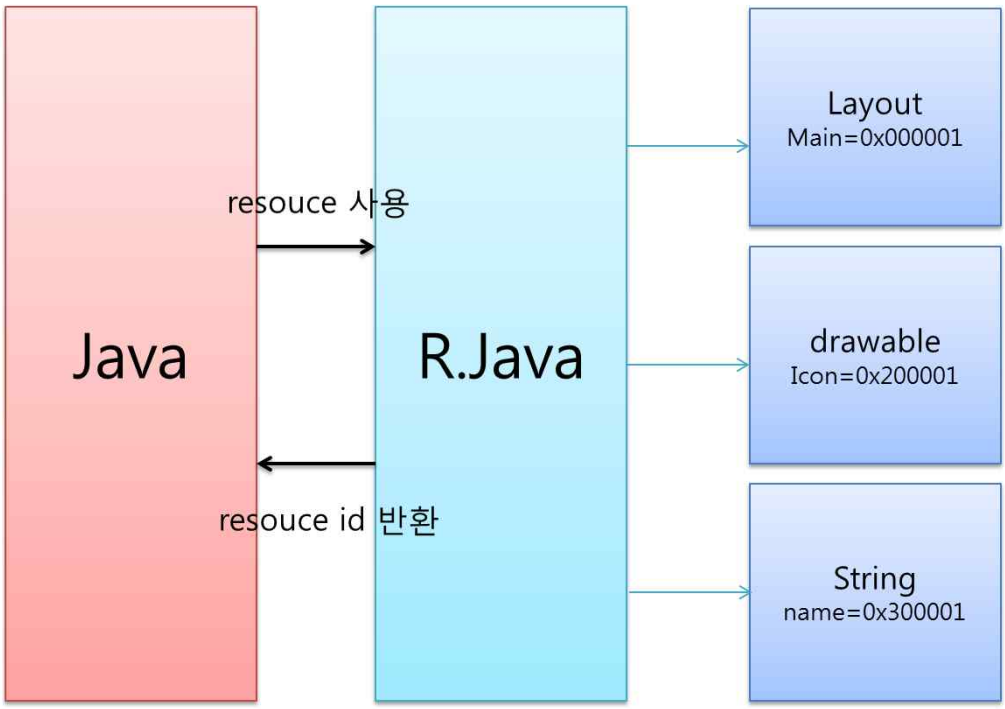
원인	현상
안드로이드 플랫폼의 구조	UI 매핑
	파라미터 처리
	비동기처리 (Async)
Java 언어적인 문제점	REST 통신 (Http)

2) 안드로이드 리소스 접근방법



[그림 25. 안드로이드 프로젝트 구조]

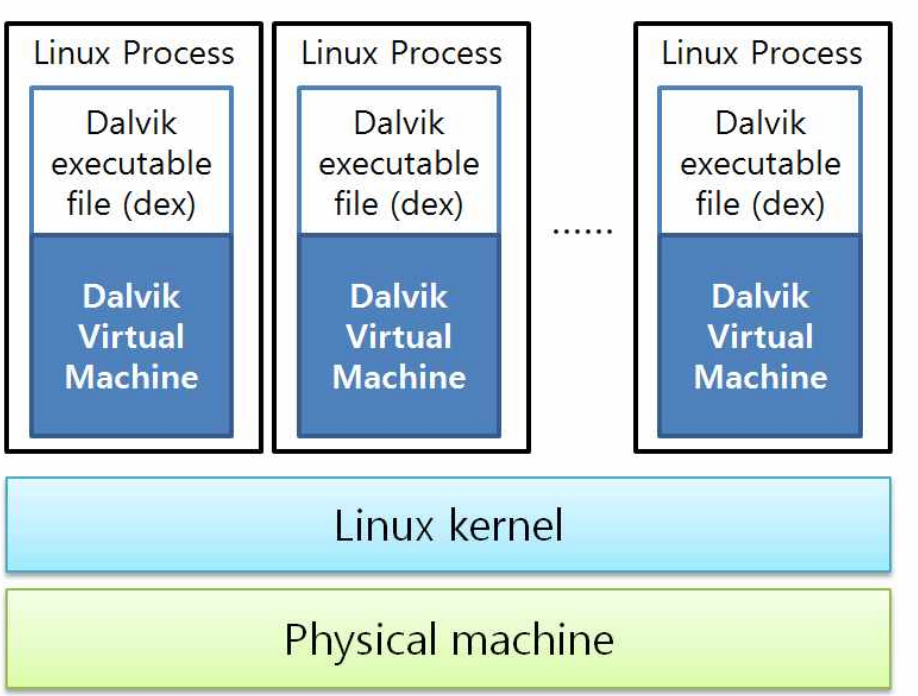
일반적인 안드로이드 어플리케이션을 개발하는 구조는 위 그림과 같이 나누어져 있다. src 폴더 하위에는 일반적으로 액티비티, 브로드캐스리시버, 서비스, 컨텐트 프로바이더가 구성이 되어있고 3rd Party 라이브러리가 함께 구성이되어 Dex(Dalvik Excutable File) 이 구성이 된다. 그리고 res 하위에는 리소스가 포함되어 Arsc(Android Package Resouce File) 이라는 파일로 묶여진다. 여기에서 가장 주목해야할 것인 AndroidManifest.xml 파일을 주목해야 한다. AndroidManifest.xml 파일은 안드로이드 어플리케이션의 설명서이자 요약서이기 때문이다. 이 모든 파일들이 모여서 APK(Android Package) 파일로 변환되어 앞서 설명한 달빅(Dalvik)가상머신 위에서 동작하게 된다.



[그림 26. 안드로이드 리소스 접근방법

안드로이드는 리소스(Resouce)를 접근 시 컴파일 할 때 자동으로 생성되는 R.java 파일을 기반으로 접근한다. 만약 리소스(Resouce)를 사용하려고 하면 R.java 에 정의된 리소스(Resouce) ID 를 획득하여 실제 리소스(Resouce)에 접근해야 한다. 실제 리소스(Resouce)를 반환하는 메소드가 바로 findViewById 이다. 외부 리소스의 이름을 직접적으로 참조하고 있지 않기 때문에 유연한 구조이지만 외부 리소스를 접근하려면 findViewById 메소드가 반복적으로 사용된다.

3) 안드로이드 데이터 전달구조(Intent)



[그림 27. 안드로이드 플랫폼 내부에서 어플리케이션 동작 구조]

위 그림과 같이 안드로이드 어플리케이션은 각각 별개의 프로세스로 동작한다. 다른 OS(윈도우즈, 리눅스) 에서는 엔트리 포인트(Entry Point)가 있으며 프로세스(Process)는 엔트리 포인트부터 시작이 된다. 하지만 안드로이드에서는 엔트리포인트를 어디서부터든지 시작할 수 있다. 이러한 아키텍쳐 구조는 어플리케이션을 확장 가능하도록 만들어 둔 것이며 개발할때에도 매우 유용하게 작용한다. 즉, A 어플리케이션의 a1,a2,a3 액티비티가 존재하고, B 어플리케이션에서 b1,b2,b3 액티비티가 있다면 A 어플리케이션에서 b1, b2, b3 액티비티를 호출 할 수 있고, B 어플리케이션에서도 a1, a2, a3 액태비티를 호출 할 수 있다. 이러한 과정을 어플리케이션 개발자는 Intent 라는 객체를 통해서 실행을 하게 된다.

```
Intent intent = new Intent(this, 어플리케이션 액티비티명);
startActivity(intent);
```

안드로이드 내부적으로 다른 어플리케이션의 액티비티를 호출하기 위해서는 다른 프로세스간의 통신 지원을 해야 하며 바인더(Binder)가 그 역할을 담당한다.

2-3-3-3. 활용할 오픈 소스 선택

앞서 코드 중복이 나타나는 원인에 대한 이유를 알아보았다. 문제는 언어적인 문제, 플랫폼적인 문제로 귀결될 수 있다. 이 문제를 해결하기 위해서 먼저 기존의 오픈 소스를 잘 활용하는 측면에서 기존에 나와 있는 몇몇 자바 기반의 IoC 프레임워크를 분석하였다.

유지보수가 쉬우려면 절대적인 코드량이 적어야 한다. 따라서, 반복적인 코드을 줄여줄 수 있어야 한다. 만일 중복된 코드를 제거하지 못한다면, 코드는 시간이 지날 수록 관리가 어렵게 되며 실제 비즈니스 로직에 집중하기가 어려워진다. '테스트 주도 개발(TDD)' 의 저자 켄트 백은 '작동하는 깔끔한 코드' 가 바로 테스트 주도 개발의 궁극적인 목표라고 말했다. 그만큼 중복된 코드는 소프트웨어가 좋은 품질을 낮추는 요소라고 할 수 있다.

가. Springframework

Springframework (http://www.springsource.org)는 복잡한 J2EE 프레임워크를 개선하기 위해서 나온 프레임워크이다. 현재는 자바진영의 많은 개발자들이 스프링 프레임워크를 사용하고 있다.
Springframework 에서는 spring-android(http://www.springsource.org/spring-android) 라는 feature 로 지원을 하지만 DI(Dependency Injection) framework 역할을 하기보다는 REST Client, OAuth 관련된 기능을 지원해주는 보조 라이브러리의 성격을 더 짙게 띄고 있다.

- A Rest Client for Android
- Auth support for accessing secure APIs

나. Guice framework

Guice(http://code.google.com/p/google-Guice/) 는 경량화된 DI(Depencency Injection) Framework 이다. 공식적으로 android 를 지원하지는 않는다.
추가적으로 roboGuice(http://code.google.com/p/roboGuice)라는 프로젝트가 안드로이드에 Guice framework 를 사용할 수 있도록 도와준다. 따라서, 안드로이드에서 Guice 를 사용할 수 있다.

하지만, 안드로이드 특성상 달빅(Dalvik) 가상머신에서 동작하기 때문에 AOP(Aspect-oriented Programming) 기능은 되지 않는다. 또한, Guice framework 는 Java 플랫폼의

표준 JSR-330 (http://jcp.org/aboutJava/communityprocess/final/jsr330/index.html) 을 지원하고 있다. JSR-330 과 Guice 지원항목을 비교하면 다음과 같다.

JSR-330 javax.inject	Guice com.google.inject	비고
@Inject	@Inject	조금 다름
@Named	@Named	교체가능함
@Qualifier	@BindingAnnotation	교체가능함
@Scope	@ScopeAnnotation	교체가능함
@Singleton	@Singleton	교체가능함
Provider	Provider	JSR-330 의 Provider 를 상속함

[표. JSR-330 과 Guice 지원항목 비교]

Guice 에서 제공하는 기능들이 더 많아서 선택적으로 사용할 수 있도록 구조화 되어있다. JSR-330 을 지원하기 때문에 JSR-330 사용한 기존 어플리케이션을 안드로이드에서 구동하기에 용이하다는 것을 의미한다.

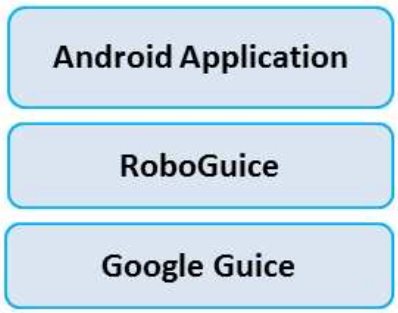
다. AndroidAnnotations

Androidannotations(http://androidannotations.org) 는 Roboguice 의 많은 기능적인 부족함을 해소할 수 있도록 도와주는 상호보완적인 프로젝트이다. 표준 스팩인 JSR269 Annotation Processing API 를 활용하여 런타임(Runtime) 시에 Code 를 인젝션(Injection) 하는 것이 아니라 컴파일(Compile) 시에 코드를 생성하는 오픈소스이다.

[그림 28. APT(Annotaion Process Tool) 진행과정]

기존 프레임워크를 분석을 진행하면서 가장 크게 비중을 두고 관찰한 점은 안드로이드의 반복적인 작업을 줄여줄 수 있고, IoC 기능을 담당하여 객체간의 의존도를 낮추는 것이 가능한가였다.

첫번째로 조사한 Springframework 는 기존 엔터프라이즈 자바시장에는 적합하나 안드로이드 플랫폼에는 단지 부가적인 기능만을 제공하는 적합하지 않은 프레임워크여서 고려 대상에는 제외하기로 했다.
두번째로 Guice framework 는 어노테이션 기반으로 쉽게 안드로이드로 확장가능하다. 안드로이드에서는 경량의 DI(Dependency Injection) Framework 를 선정하는 것이 더 바람직하다고 판단하였고, JSR-330 표준을 지원하는 Guice framework 를 안드로이드에 적용할 수 있는 RoboGuice 를 적용하기로 하였다. RobuGuice 는 일반적인 객체를 인젝션(Injection) 할 수도 있지만 안드로이드에 특화된 객체들도 인젝션 가능하다. Roboguice 로 안드로이드 어플리케이션을 개발한다는 것은 다음과 같은 기술 스택으로 어플리케이션을 개발하겠다는 것이다.

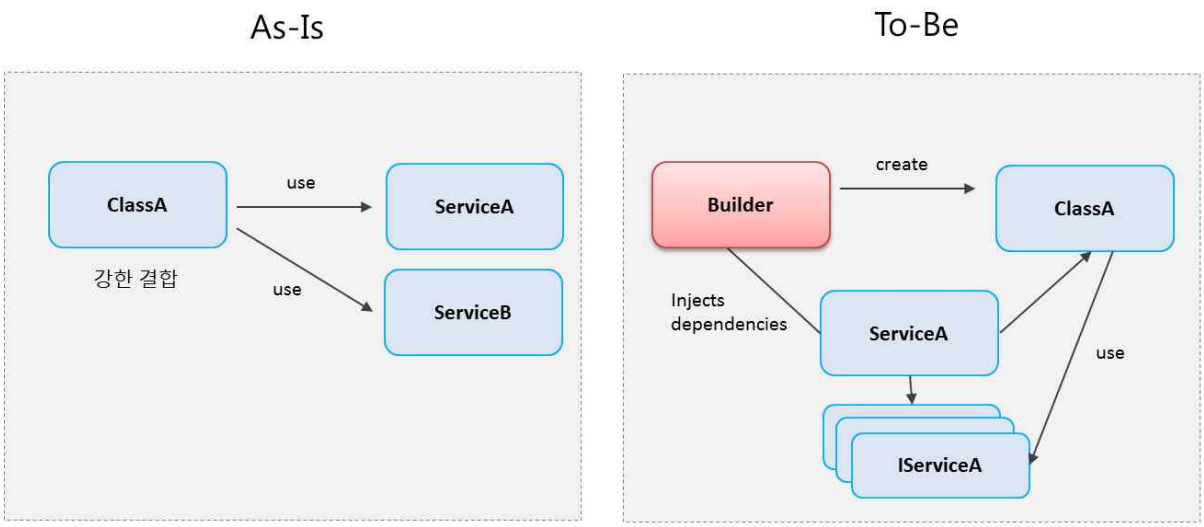


[그림 29. Roboguice 활용하여 어플리케이션 개발 시 아키텍쳐]

세번째로 Androidannotations 는 Roboguice 가 단점으로 지닌 성능상의 이점을 가지고 있다. 또한, 안드로이드에 특화된 DI(Dependency Injection) 기능들을 더 많이 제공하기 때문에 RoboGuice 와 함께 적용한다면 더 많은 중복코드를 제거할 수 있을 것이라고 판단하였다.

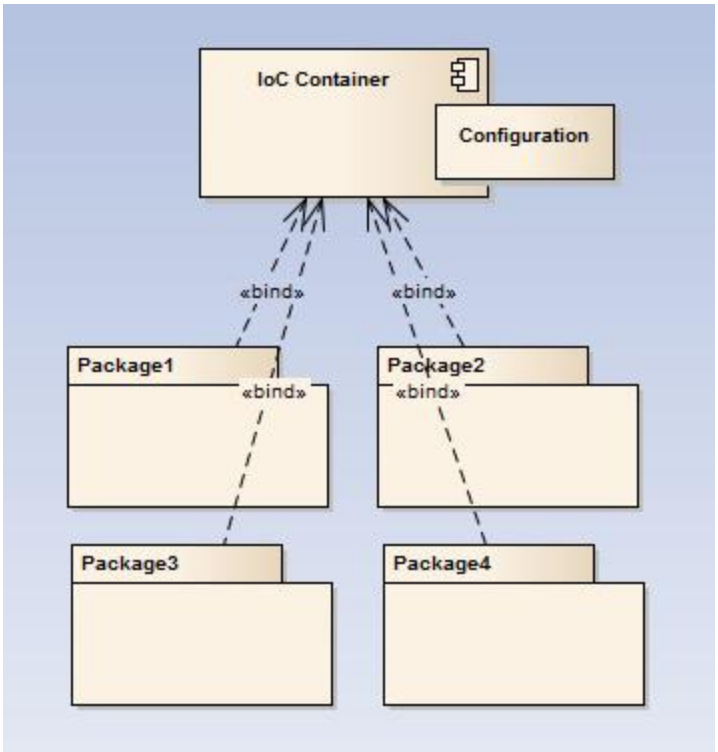
2-3-4. 아키텍쳐 개선 활동

보다 구조적이고 solid 한 아키텍쳐를 만들기 위해서, 기존 구조를 분석해서 문제점을 파악하여 간단한 구조부터 만들고자 했다.



[그림 30. Roboguice 활용하여 어플리케이션 개발 시 아키텍쳐]

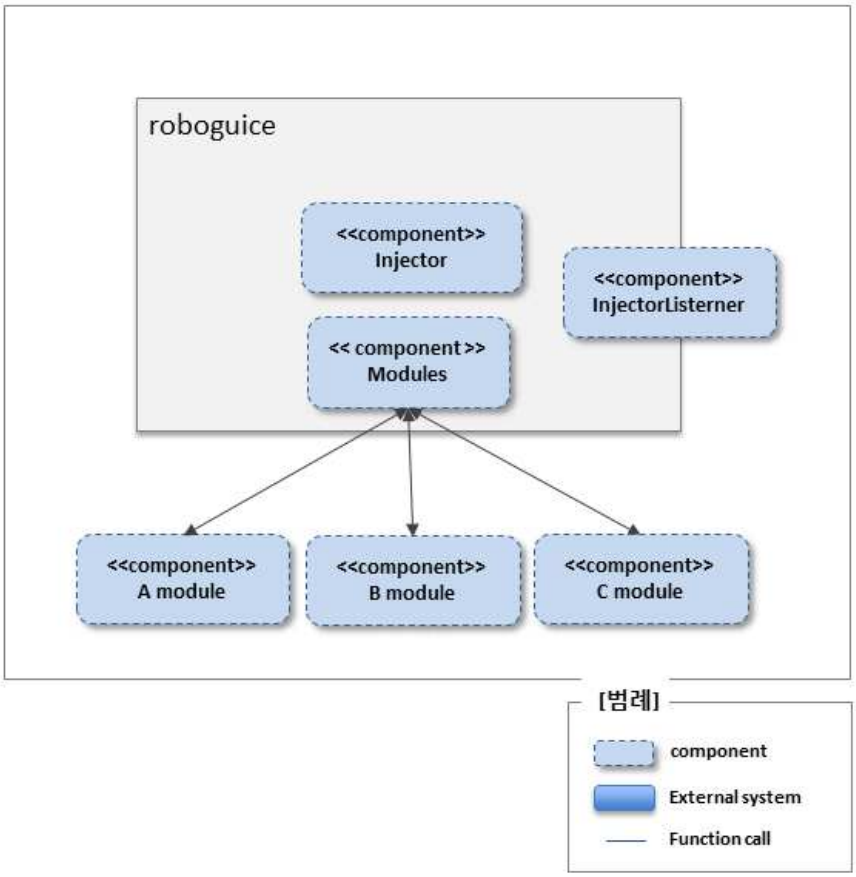
또한, IoC pattern 을 이용하여 문제를 바라보기로 하였다. IoC 는 객체간의 관계를 느슨하게 만들어 변화에 유연하게 만들어 좀 더 쉽게 사용할 수 있도록 해주는데 목적이 있는 패턴이다.



[그림 31. IoC 패턴]

객체간의 의존도를 낮추기 위해서 IoC pattern 을 적용한 프레임워크를 적용하여 유지보수성이 높은 패턴을 적용하여 이 문제를 해결했다.

2-3-5. 아키텍쳐 개선 결과



[그림 32. IoC 프레임워크 아키텍쳐]

기존에는 객체와 객체간의 결합도가 높아 변경에 대해서 유연하게 대처하지 못하는 아키텍쳐였다.

이러한 문제를 해결하기 위해, 기존 오픈소스 RoboGuice 를 적용하여 구조가 유연하게 변경되었다. 주요 컴포넌트들은 다음과 같다.

- Module: 일반적인 Module 레벨의 인터페이스이다. 실제 코드는 AbstractModule 을 상속하여 구현한다.

- Injector: 개발자가 모듈별로 나누어서 객체를 정의하였다면 여기에서는 객체를 로드하여 실제로 조립하는 과정을 진행한다.
- InjectorListner: 인젝션을 진행하면서 나오는 결과들을 리스너를 통해서 알려줄 수 있다.

Guice 는 런타임시에 어노테이션을 읽어 다른 객체에 인젝션(Injection) 하는 과정을 거친다. 내부에서 타입과 클래스의 이름을 기반으로 판별하여 해당 객체에 인젝션할지를 판단하는 과정을 위 그림처럼 표현할 수가 있다. 실제 구현자의 입장에서의 코드는 다음과 같이 구현된다.

```
import com.google.inject.AbstractModule;

public class MyModule extends AbstractModule {
  @Override
  protected void configure() {
        bind(Person.class);
  }
}
```

앞서 얘기한 Module 은 다음과 같이 코드가 작성된다. AbstractModule 이라는 Guice 에서 제공하는 작은 단위로 정의를 하고 바인딩(binding) 할 객체들을 등록해주면 끝이다. 내부에서는 리플렉션(Reflection)을 사용하여 바인딩된 객체를 타입캐스팅하고 인젝션(Injection) 하는 과정들을 거친다. 안드로이드 어플리케이션을 개발 시 Roboguice 관점에서는 다음과 같은 구조로 구성되게 된다. 다양한 Module 들이 구성이 되고, 이 모듈들이 구성되어 하나의 어플리케이션이 이루어진다.

2-3-5-1. 개발 생산성 개선

가. UI 매핑

안드로이드 어플리케이션을 개발하면서 UI 관련된 구성을 하기 위해서는 많은 설정파일이 필요하다. 만약, UI 위젯들이 많은 화면을 구성하려고 한다면 위와 같은 코드가 계속적으로 반복이 될 것이다. 즉 findViewById 라는 간단한 코드가 보일러(Boiler) 코드가 된다는 것이다.

기존 방법

```
TextView subject = (TextView) findViewById(R.id.subject);
TextView writer = (TextView) findViewById(R.id.writer);
TextView date = (TextView) findViewById(R.id.date);
TextView hit = (TextView) findViewById(R.id.hit);
```

개선 후,

```
@ViewById TextView subject;
@ViewById TextView write;
@ViewById TextView date
@ViewById TextView hit;
```

위의 코드처럼 쉽게 어노테이션만 설정하면 반복되는 코드들을 줄여줄 수 있다.

나. 파라미터 처리

안드로이드에서는 액티비티(Activity) 라는 것으로 화면을 구성하고 이에 따른 동작을 코드로 구현을 한다. 만약 게시판을 만든다고 가정을 해보자. 게시물 목록에서 게시 글을 터치하고 다음화면을 이동을 하길 원할 것이다. 즉 액티비티(Activity)와 액티비티(Activity) 간의 데이터흐름이 일어난다는 뜻이다. 안드로이드 플랫폼에서는 이러한 데이터 흐름을 인텐트(Intent) 라고 추상화하여 지정을 하고 그에 맞는 데이터는 액티비티(Activity) 내에서 값처리를 해야한다. 즉, 다음과 같은 코드들이 들어간다는 뜻이다. Id, name, nickname sex, object 타입에 맞게 전달된 데이터를 불러와야 하며, 값이 정상적으로 전달이 되었는지도 해당 내부 액티비티(Activity) 내에서 처리를 해야한다. 단지, A 화면과 B 화면의 데이터를 전달하기 위해서 개발자는 getStringExtra 같은 반복되는 코드를 구현해야 한다.

기존 방법

```
String id = intent.getStringExtra("id");
String name = intent.getStringExtra(“name");
String nickname = intent.getStringExtra(“nickname");
int sex = intent.getIntExtra(“sex“,0);
Object object = intent.getExtras().get("object");
```

개선 후

```
@Extra(“id")   String id;
```

```
@Extra(“name")   String name;
@Extra(“nickname") String nickname;
@Extra(“sex") int sex;
@Extra(“object") Object object;
```

개선 후에는 파라미터의 타입에 맞게 캐스팅(Casting)을 하거나 검증할 필요가 없다. 자동으로 개선전의 작업을 어노테이션으로 대체하고 있다.

다. 비동기처리 (Async)

사용자들은 빠른 반응속도의 애플리케이션을 원하는데 만약 특정 애플리케이션에서 2 초 이내에 처리 반응이 일어나지 않으면 사용자는 느리다라고 느끼고 애플리케이션에 불평을 토로한다. 따라서, 안드로이드 플랫폼에서는 이러한 사용자의 불만을 줄이고 원활한 시스템을 운용하기 위해서 일정 시간 동안 응답하지 애플리케이션을 감시하다가 사용자에게 다이얼로그를 나타내어 계속 동작하게 할 것인지 강제종료를 할 것인지 선택할 수 있도록 한다. 안드로이드에서는 ANR(Application Not Responding) 다이얼로그를 사용자에게 노출한다. 따라서, 데이터를 파일에 저장한다든지 네트워크를 통해서 데이터를 다운로드, 업로드 하는 경우에는 반드시 비동기처리를 해야한다. 비동기처리를 하는 코드를 보면 다음과 같다.

개발자는 단순히 비동기처리를 하고 싶은 것인데 관리에 필요한 반복적인 코드를 만들게 된다.

기존 방법

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) {
```

개선 후

```
@Background
void backgroudJob() {
MovieContents movieContents =
daumMovieService.getMovieContents("love");
String result = movieContents.getQuery();
setData(result);
}
@UiThread
void setData(String data) {
textView.setText(data);
}
```

안드로이드에서는 비동기처리를 위해서 별도 쓰레드를 생성하거나 AsyncTask 를 활용하여 구현하는데 어노테이션을 위와 같이 설정하면 자동으로 쓰레드를 만들어 메소드의 내용을 실행하는 구조로 변경이 되었다. 이렇게 되면서 실제 비즈니스로직에만 집중할 수 있게 되어 반복적인 작업들이 줄어들게 된다.

라. REST 통신 (Http)

안드로이드에서 Http 통신을 하기 위해서는 아래와 같은 복잡한 코드를 구현해야 한다. 실제 비즈니스로직보다는 불안정한 네트워크일 때의 처리를 위한 예외적인 상황에 대한 구현이 더 많이 들어간다.

기존 방법

```
private InputStream download(String url) {
HttpURLConnection con = null;
URL url;
InputStream is=null;
try {
  url = new URL(url);
  con = (HttpURLConnection) url.openConnection();
  con.setReadTimeout(10000 /* milliseconds */);
  con.setConnectTimeout(15000 /* milliseconds */);
  con.setRequestMethod("GET");
```

```
con.setDoInput(true);
```

개선 후

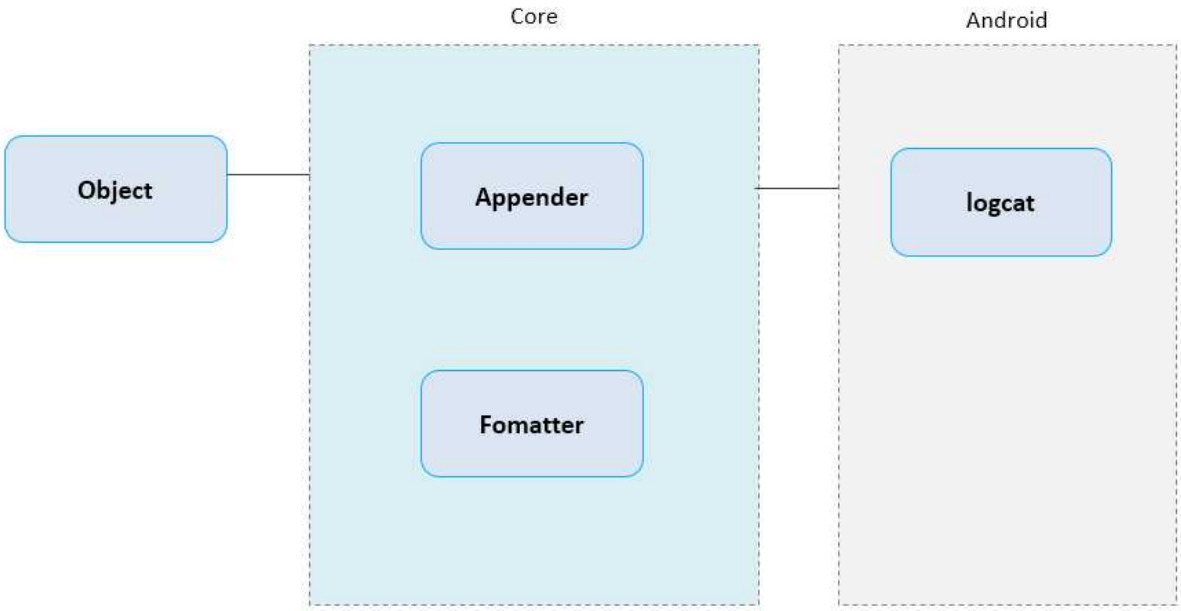
```
@Rest
public interface DaumMovieService {
@Get("http://apis.daum.net/contents/movie?
apikey=DAUM_CONTENTS_DEMO_APIKEY&output=xml&q={query}")
public MovieContents getMovieContents(String query);
}
```

RESTful 하게 연결될 인터페이스만 정의하고 그에 맞는 파라미터를 정의하면 실제로 이 코드를 자동으로 네트워크 연결 그리고 데이터 마셜링까지 자동으로 이루어진다. 실제로 몇백줄의 코드를 단 몇줄로 줄여줄 수 있는 것이다.

2-4. Logging framework - logdog

2-4-1. 개요 및 배경

안드로이드(Android) 어플리케이션(Application)을 개발하다 보면 오류가 발생하고 해당 오류를 확인하기 위해서 디버깅을 수행하거나 로그를 확인한다. 로그는 시스템이나 응용 프로그램의 동작에 대한 상세한 기록이다. 안드로이드 시스템은 장비에서 발생하는 일들에 대해서 로그를 남기며 응용 프로그램도 디버깅이나 실행 흐름 추적등을 위해 수시로 로그를 남길수 있다. 디버깅의 경우 느리고 직접 디버깅을 해봐야 프로그램의 상태를 파악할수 있는데 비해 로그는 항상 출력되며 실시간으로 프로그램의 상태를 출력할수 있어 유용하다.



[그림 33. Android 의 logging framework]

안드로이드에서는 위의 구조처럼 로그 확인을 IDE 툴(eclipse)의 DDMS(Dalvik Debug Monitor Service)의 LogCat 탭 또는 ADB(Android Debug Bridge) 툴로 명령행에서 확인할수 있다. LogCat 은 에뮬레이터 또는 기기 상태와 관련된 로그를 출력한다. 만일 런타임 에러가 발생한 경우에는 구체적으로 어디에서 문제가 발생했는지 출력된다. 또한 스텍 트레이스 같은 것도 포함하고 있어, 이 로그를 통해 프로그램에 어떤 일들이 일어나는지를 알 수 있다. 이런 기능 때문에 개발자는 LogCat 로그로 오류를 확인할 수 있다.

LogCat 은 파일로 저장되지 않아 이클립스 에서만 로그를 볼 수 있기 때문에 효율적인 로그 관리가 어렵다. 물론 LogCat 을 파일로 저장하는 방법이 존재한다. 예를 들어 콘솔창을 이용해 텍스트 파일로 저장하는 코드를 직접 입력하는 방식이다. 그러나 이 방법은 코드를 직접 만든 이클립스가 아닌 콘솔창에 Log Cat 을 저장하는 코드를 매번 직접 입력해야 하는 번거로움이 있다. 그렇기 때문에 배포된 안드로이드 어플리케이션에 심각한 오류가 발생했을 경우, LogCat 을 확인하기 위해서 고객의 폰을 직접 확인해야 하기 때문에, ERROR 이나 FATAL 레벨(level)의 로그를 지속적으로 모니터링 하는 것은 거의 불가능하다. 그리고 안드로이드 시스템내의 많은 어플리케이션에서, 다양한 로그 메시지들을 유발하기 때문에, 이중 다른 어플리케이션에서 발생한 메시지들을 식별하고 분리하는 작업도 매우 어려운 단점이 있다.

그리고 Android 의 logging framework 는 6 개의 logging level 이 존재하지만, Android SDK 의 Log 출력방식은 Verbose, Debug 는 상황에 따라서 출력되지만, Warning, Information, Error 의 경우에는 항상 출력되어 내가 원하는 logging level 만 logcat 에서 확인할수 있도록 할수 없다는 문제점이 있다.

2-4-2. 설계 개선 목표

Logging framework 에 대해서 LogCat 의 문제점을 보안하고 로그 관리에 활용할 수 있도록 설계를 개선하고 refactoring 하기로 했다.

즉, 다양한 Appender 를 제공하여 개발할 때뿐만 아니라 애플리케이션을 릴리즈한 상태에서도 사용할수 있고, 기기의 SD 카드에 로그를 저장하거나, 웹 서버로도 로그 정보를 수집할수 있도록 하였다. 즉, 오류 발생 시 해당 오류가 발생한 로그를 확인하고 매일 쌓여가는 엄청난 로그양을 체계적으로 관리해 프로그램을 효율적으로 개발할 수 있도록 하였다.

2-4-3. 아키텍쳐 개선을 위한 사전 활동

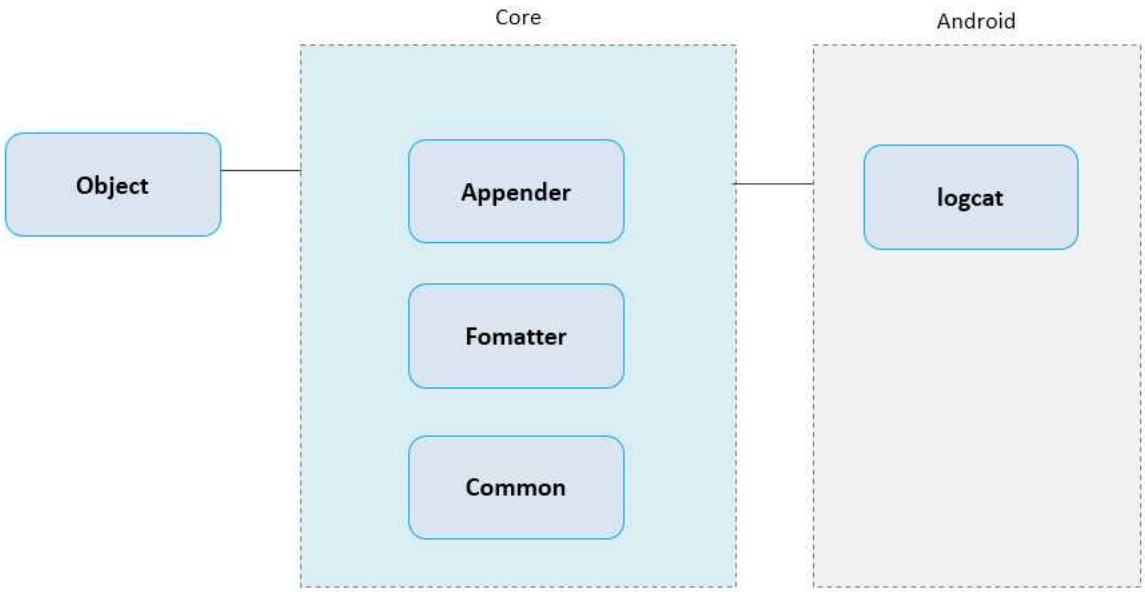
2-4-3-1. 아키텍쳐 개선의 시작

본 장에서는 logging framework 아키텍쳐 개선 작업을 위하여 기존 문제점을 해결하고 사용성이 높은 아키텍쳐를 구현하고자 한다.

2-4-3-2. 활용할 오픈 소스 선택

먼저 기존의 오픈 소스를 잘 활용하는 측면에서 기존에 나와있는 MicroLog4Android 를 조사하였다. MicroLog4Android 는 안드로이드용 로그 오픈소스 라이브러리로서

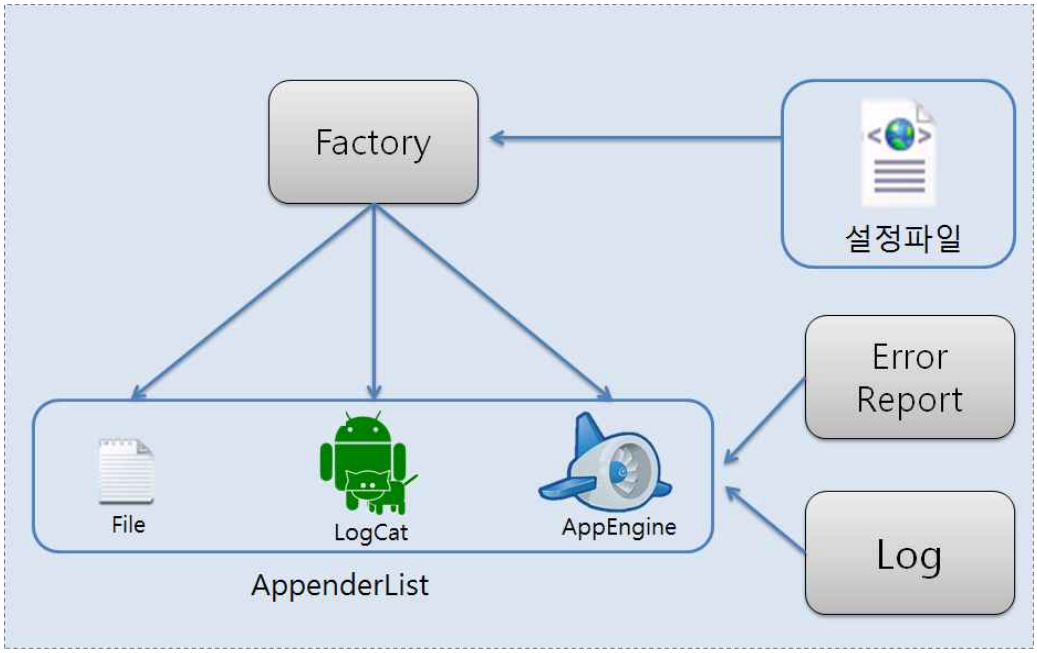
LogCat 에 보여줌과 동시에 파일로도 저장이 가능하지만, 로그정보를 원격서버로 보내는 기능이 없고 소스 코드단에서 Appender 와 Formatter 를 설정해야 해서 설정값을 변경하고자 하면 소스코드를 변경 해야하는 한계가 있다.



[그림 34. MicroLog4Android 의 아키텍쳐]

하지만, MicroLog4Android 라이브러리는 LogCat 의 단점을 보완한 가장 대표적인 오픈소스 라이브러리 이며, 이미 많은 사람들이 사용하고 있어서 검증이 되었다는 장점이 있어서, 본 문서에서는 MicroLog4Android 오픈소스를 fork 한 후 리팩토링 해서 더 좋게 개선하고자 하였다.

2-4-3-3. 사용성 개선을 위한 방향성 도출



[그림 35. 사용성 개선의 기본 방향]

사용성 개선은 개발자가 쉽게 File, LogCat, 웹서버를 통해서 logging 할수있도록 하는 것을 최우선 목표로 해서 기본개념을 설정하고, Appender 와 Formatter 의 설정을 소스코드가 아닌 설정파일을 통해서 할수있도록 흐름을 도출해 나갔다.

```
public class LogtestActivity extends Activity {
    static public Logger logger = LoggerFactory.getLogger();
        @Override
        public void onCreate(Bundle savedInstanceState) {

            PatternFormatter formatter = new PatternFormatter();
            formatter.setPattern( " %d{ISO8601} [%P] %m %T " );
            logger.setLevel(Level.INFO);

             // write to LogCat
            LogCatAppender logCatAppender = new LogCatAppender();
            logCatAppender.setFormatter(formatter);
            logger.addAppender(logCatAppender);
```

[그림 36. Log4Android 의 로깅 방법 및 Appender 와 Formatter 설정]

```
public class LogDogClient extends Application {
@Override
public void onCreate() {
// TODO Auto-generated method stub
   super.onCreate();
   LogDog.LogDoginitialize(this, " 설정.xml");
   }
}
```

[그림 37. logdog 의 로깅 방법]

```
<Log>
   <Level>debug</Level>
</Log>

<Appenders>
   <AppenderList class="java.util.ArrayList">

      <Appender AppenderName="LogCat">
      <Formatter class="com.logdog.Formatter.PatternFormatter">
            <PatternString>   %d{ISO8601}   [%P]  %m  %T  </PatternString>
      </Formatter>
      </Appender>

       </AppenderList>
   </Appenders>
```

[그림 38. logdog 의 Appender 와 Formatter 설정]

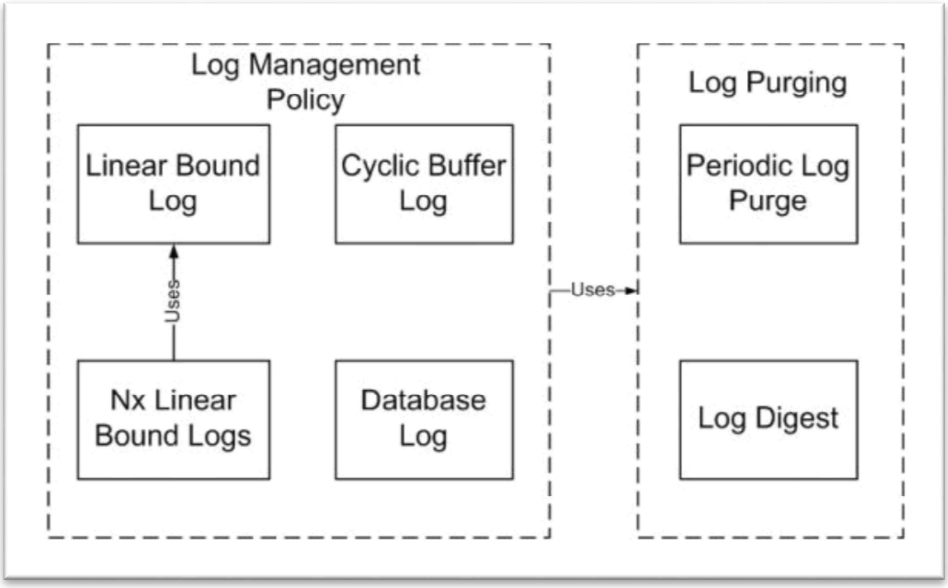
2-4-3-4. microlog4android

microlog4android 에 대한 내용을 알아보기 전 로그 데이터를 어떻게 효율적으로 관리할 수 있을지 살펴본 다음 microlog4android 를 좀더 자세히 살펴보겠다.

가. 로그 관리 패턴(Log Management Pattern)

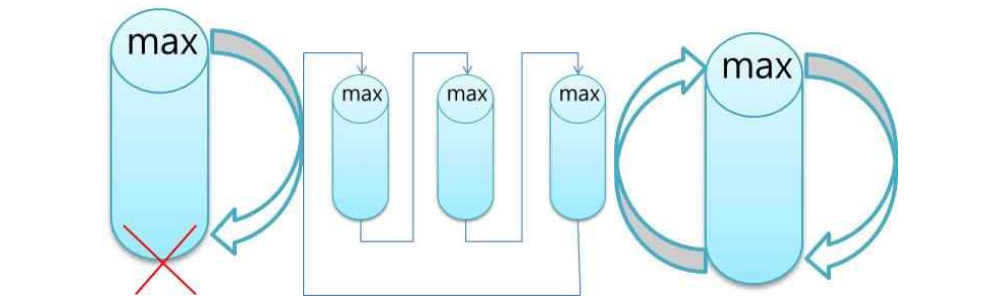
수집된 수 많은 로그들을 어떻게 관리해야 편리하게 사용할 수 있을까? 또 어떻게 관리해야 Overflow 가 발생하지 않을까? 그리고 수 많은 로그 중 중요한 로그만 남기고 싶을 때는 어떤 방법을 사용해야 할까? 소프트웨어를 사용하다 실패하면 우리는 무엇이 어디서부터 잘못됐는지에 대한 정보를 로그에서 얻게 된다. 하지만 이런 로그들은 하루에도 엄청난 양이 쌓이게 된다. 이렇게 매일 쌓인 로그들은 어느 시점에 이르면 수집할 여유 공간이 없어져 중요한 로그들이 삭제되고 관리가 점점 힘들어진다. 그래서 로그를

나중에라도 참조할 수 있도록 유지하는 매커니즘을 설계해야 한다. 이런 메커니즘을 설계하기 위해 로그 관리 패턴들을 하나씩 살펴보겠다.



[그림 39. 로그 관리 패턴]

<그림 1>에서 보는 것처럼 기본적인 로그 패턴은 Linear Bound Log, Cyclic Buffer Log, Database Log 그리고 Linear Bound Log 를 응용한 Nx Linear Bound Logs 가 있다. 또 이 네 가지 패턴을 모두 활용한 Periodic Log shrink 과 Log Digest 이 있다. 이 로그 패턴들이 언제 사용되는지 지금부터 살펴보자.



[그림 40. Linerar Log 와 Cyclic Log]

Linear Bound Log 는 로그 관리 패턴의 가장 간단한 방법으로 로그의 최대 사이즈를 할당하는 방법이다. 로그를 기록하다가 최대 사이즈에 도달하면 다시 로그 공간을 재활용해 처음부터 다시 로그를 기록한다. 이 방법은 로그양이 제한돼 있으므로 중요한 사건이

발생한 데이터가 언제 저장될지 모르고, 해당 사이즈가 넘어가면 이전 데이터들이 모두 삭제되므로 로그 분석에는 한계가 있다.

Nx Linear Bound Log 는 Linear Bound Log 을 보완해 다수의 Linear Bound Log 를 만드는 방법이다. 하나의 로그가 최대 사이즈에 도달하면 새로운 로그를 생성하고(Linear Bound Log 정해진 개수만큼), 모든 Linear Bound Log 을 사용하면 다시 처음 로그부터 재사용하는 방법이다.

Cyclic Buffer Log 는 Linear Bound Log 를 보완해 최대 사이즈까지 도달하면, 로그 파일을 다 지우지 않고 처음부터 덮어 쓰는 방법이다. 가장 이상적인 방법이며 향상된 기능으로 Daily Rolling 을 제공한다. 이 방법을 사용하려면 저장된 로그를 읽을 때, 오래된 이벤트 위치를 찾아 해당 위치부터 읽기 시작해야 한다.

Database Log 는 파일 대신 데이터베이스에 남기는 방법이다. 로그를 액세스하고 불순물을 제거하는 퍼지(Purge)를 수행하기 쉽다. 하지만 많은 Processing Overhead 가 요구된다.
소개한 로그 관리 정책들을 모두 적용할 수 있는 로그 퍼징 패턴들도 있다.
Peoridic Log shrink 는 이벤트 또는 로그를 삭제해 로그 축소 주기를 실행하는 방법이다. 가장 기본적으로 미리 정의된 스케줄러로 로그에서 이벤트를 삭제하는 코드를 생성하기 위한 '스케줄링 메커니즘'을 사용해, 보조 스토리지에 대한 로그와 이동 이벤트 로그를 삭제한다.

Log digest 는 로그에 대한 이벤트 요약을 생성해 로그 퍼지 시 손실을 줄일 수 있는 방법이다.

나. microlog4android

microlog4android 라이브러리는 강력한 오픈소스 로깅 툴(Logging Tool)이다. microlog4android 에서는 Log4j API 를 사용한다. 자바 사용자라면 누구나 Log4j 를 많이 사용하기 때문에 쉽게 microlog 4android 에 접근할 수 있다.
다음은 microlog4android 라이브러리의 장점이다. 특히 로그 정보를 파일로 저장하거나, 특정 서버로 전송할 수 있다는 장점은 개발 시 많은 편의를 제공한다.

① SL4J API
microlog4android 로 SL4J API 를 사용할 수 있다. 곧 다른 로깅 구현을 바꿀 수 있는 자유가 제공된다는 것이다. 예를 들어 파일 로깅을 위해 개발 시에는

microlog4android 를 사용하고, 배포 시 로깅을 제거하고 싶다면 microlog4android jar 와 NOP 구현을 교체하면 된다. 또 Android LogCat logging 으로 되돌리고 싶을 때는 SLF4J 구현을 사용하면 된다.

② File logging

Android logging classes 를 사용해 기기의 SD 카드에 로깅할 수 있는 방법은 없다. 그러나 microlog4android 는 SD 카드에 로그를 저장할 수 있다. 이렇게 파일로 저장된 로그들을 가지고 로그 관리 패턴을 적용하면 더 효율적으로 로그를 관리할 수 있다.

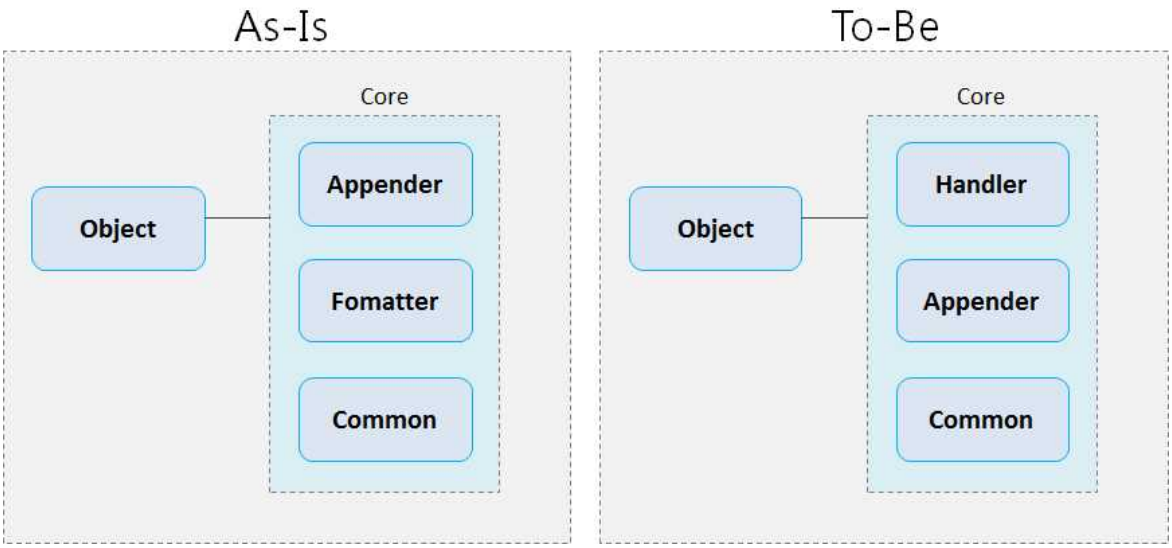
③ Remote logging

microlog4android 는 microlog 와 같은 원격 로깅을 지원한다. 그래서 수동으로 로그 파일을 전송할 필요가 없고, 로그가 발생하면 특정 서버로 전송해 자동으로 로그를 기록할 수 있다.

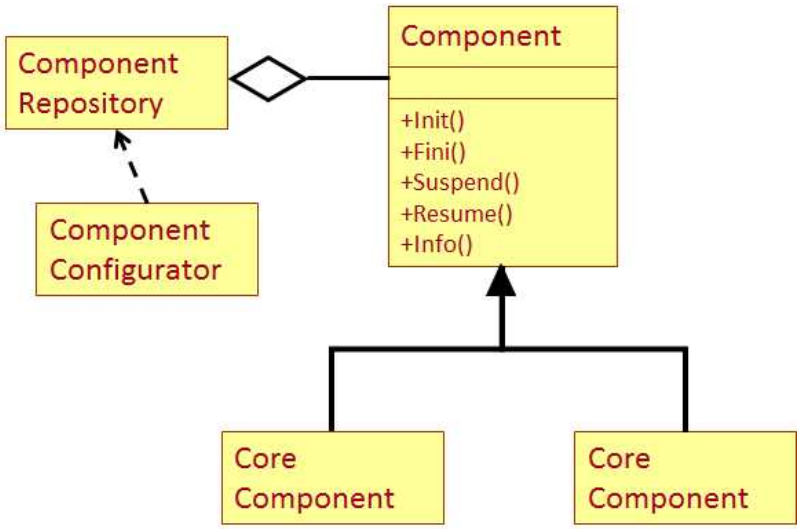
또 소프트웨어에 오류가 발생했을 때도 해당 소프트웨어가 설치된 곳까지 가서 로그를 확인하지 않고, 개발한 곳에서 확인해 오류를 수정할 수 있다.

2-4-4. 아키텍쳐 개선 활동

보다 구조적이고 solid 한 아키텍쳐를 만들기 위해서, 기존 구조를 분석해서 문제점을 파악하여 간단한 구조부터 만들고자 했다.



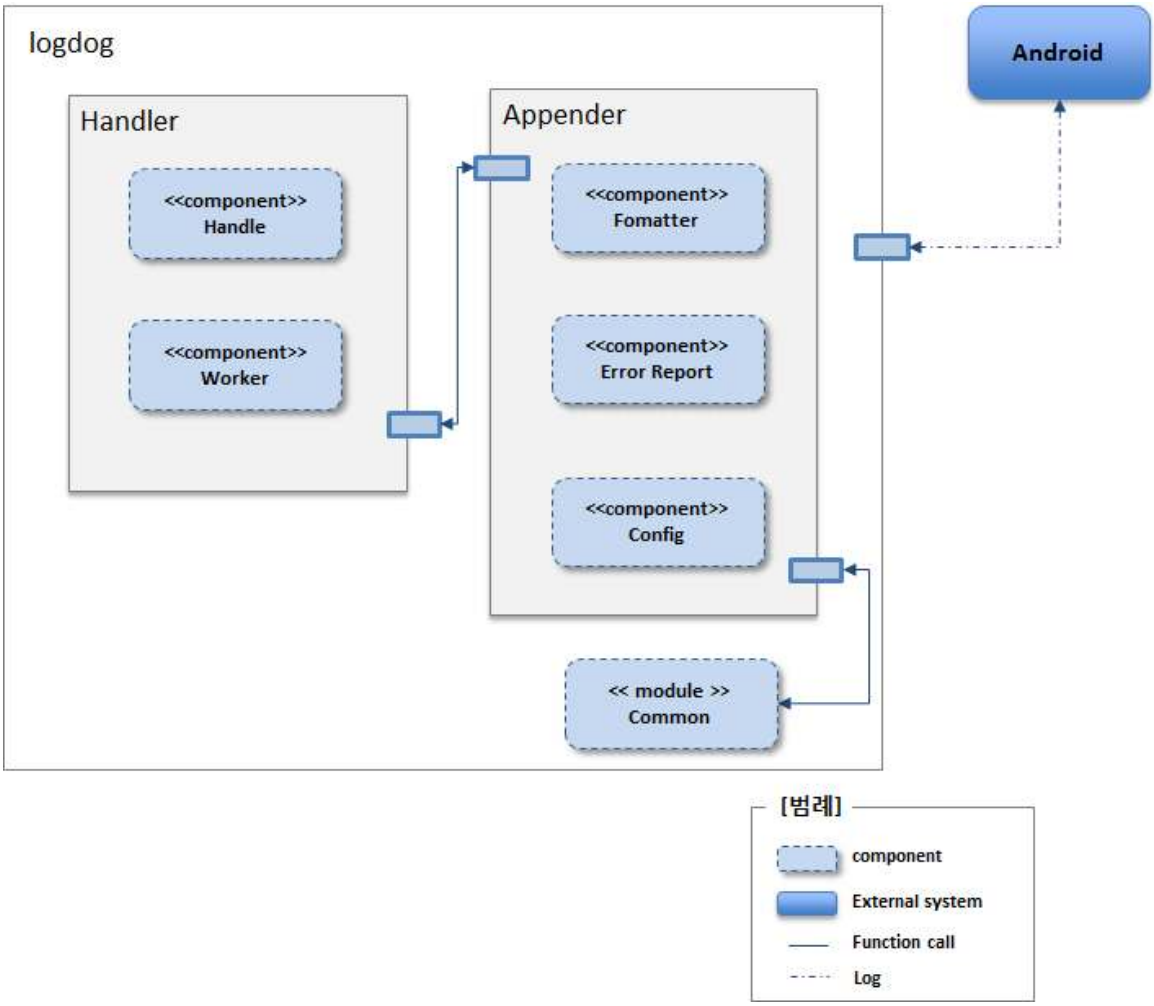
[그림 41. 개선된 logging 프레임워크 아키텍처]



[그림 42. Component Configurator 패턴]

본 문서에서는 Component Configurator 패턴을 이용하여 MicroLog4Android 를 리팩토링하였다. Component Configurator 패턴은 런타임시 사용하는 컴포넌트들을 제어하고, 시스템을 중지 없이 컴포넌트를 추가, 제거, 교체 할수 있는 패턴이다. 이렇게 함으로써 런타임시에 설정파일을 통해서 Appender 와 Formatter 를 설정 함으로써 변화에 유연하게 대처할수 있다.

2-4-5. 아키텍쳐 개선 결과



[그림 43. Component Configurator 패턴]

기존 오픈소스 MicroLog4Android 를 캡슐화하여 다음과 같이 컴포넌트들을 구성하였다.

- Handler: logging 에서의 모든 과정을 전담하며, 개발자가 확인하지 못한 Exception 도 Catch 해서 처리한다.
- Appender: File, LogCat, AppEngine Appender 가 세팅파일에 의해서 생성되며, Error Report 로그를 작동하고, Fomatter 설정값대로 Log 메시지를 생성 하는 역할을 한다.
- Common: Date, Network, Json Parsing 과 관련된 기능들을 제공한다.

2-4-5-1. 개발 생성성 및 관리 용이성 개선

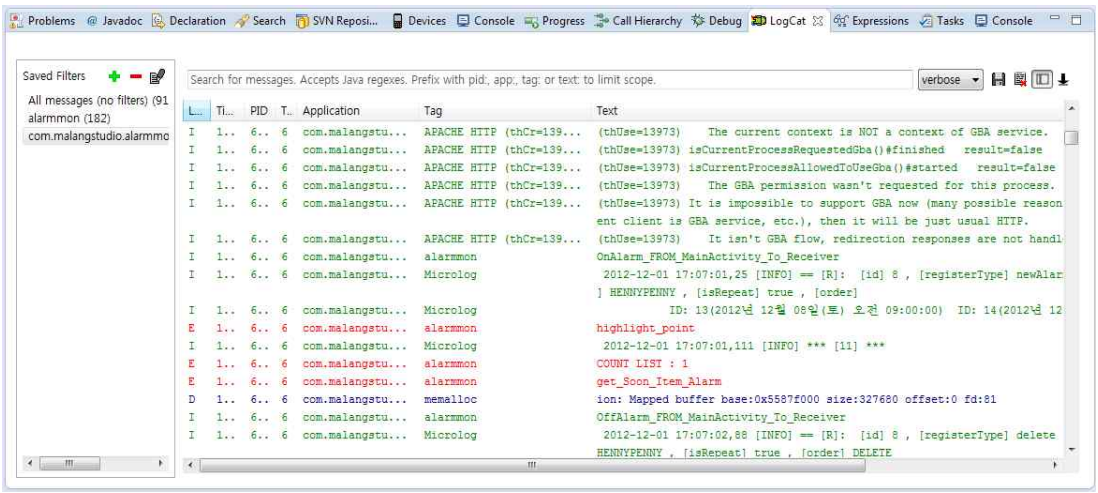
다음과 같은 부분에서 개발 생산성 및 관리 용이성 개선이 이루어졌다.

가. LogCat 출력

- 기존 LogCat 방법

```
Log.d("tag 명", "Debug 로그 메세지");
Log.e("tag 명", "Error 로그 메세지");
Log.w("tag 명", "Warning 로그 메세지");
Log.i("tag 명", "Information 로그 메세지");
Log.v("tag 명", "Verbose 로그 메세지");
```

위에서 Verbose 로그는 개발중에만 컴파일 되며 Debug 로그는 실행중에는 제거된다. 나머지 세개의 로그는 항상 유지되는데 심각한 에러 상황이면 Error 로그를 사용하고 경미한 경고일땐 Warning, 일반적인 정보라면 Information 로그를 사용한다.



[그림 44. Android DDMS 의 LogCat]

위 그림은 Android SDK 의 Log 클래스를 사용하여 LogCat 으로 로그메시지를 출력한 예이다.

- logdog 구현 방법

```
LogDog.Print(Level.DEBUG, "DEBUG 로그 메세지");
LogDog.Print(Level.INFO, "INFO 로그 메세지");
LogDog.Print(Level.ERROR, "ERROR 로그 메세지");
LogDog.Print(Level.FATAL, "FATAL 로그 메세지");
LogDog.Print(Level.WARN, "WARN 로그 메세지");
```

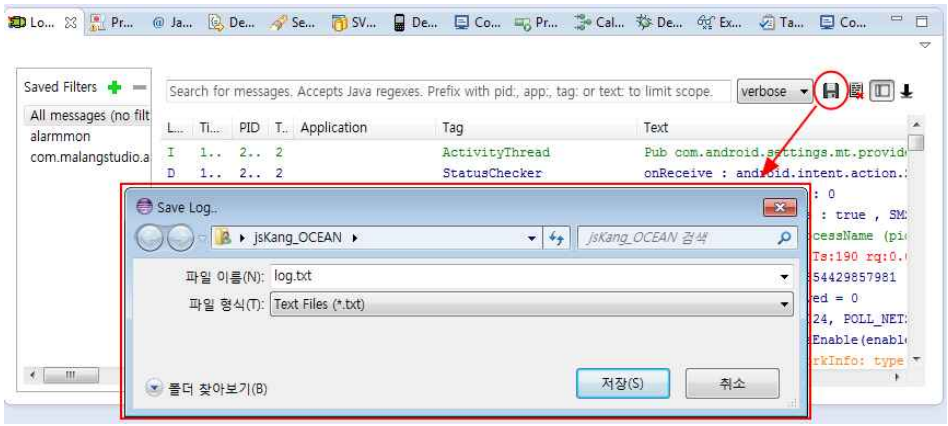
LogDog 의 정적메소드인 Print 를 통해서 fatal, error, warn, info, debug, trace 레벨별로 로깅할수 있다. 이때 Logger 객체의 레벨보다 낮은 레벨을 갖는 로그메시지는 출력되지 않는다. 이때 설정파일에 LogCatAppender 가 추가되었다면 로그메시지를 LogCat 에서도 확인할수 있다. 그리고 로그 출력패턴을 " %d{ISO8601} [%P] %m "로 설정하였을 경우, 다음과 같은 형태로 로그메시지가 출력되기 때문에 Android SDK 에서 기본 제공되는 로그 출력방식 보다 더 효율적으로 로그 메시지를 출력할 수 있다. 그리고 디폴트 레벨로 INFO 를 설정했을 경우, 더 낮은 레벨인 DEBUG, TRACE 를 갖는 로그메시지의 경우 출력되지 않는다.

```
2012-11-27  13:01:22,139  [FATAL] FATAL 로그 메시지
2012-11-27  13:01:22,140  [ERROR] ERROR 로그 메시지
2012-11-27  13:01:22,141  [WARN] WARN 로그 메시지
2012-11-27  13:01:22,142  [INFO] INFO 로그 메시지
```

이는 매우 중요한 개념으로 이어진다. Android SDK 의 Log 출력방식은 Verbose, Debug 는 상황에 따라서 출력되지만, Warning, Information, Error 의 경우에는 항상 출력된다. 하지만, logdog 의 경우 외부 환경설정 파일을 만들어서 로그 출력 패턴, 파일명, Logger 의 레벨등을 지정해두고, Logger 객체 생성시 이 환경설정 파일을 참조한다면, logdog 의 설정내용 변경시, 소스를 고치고 다시 컴파일 할 필요 없이 Logger 의 level, 로그 출력 패턴, 파일명 등을 외부에서 변경가능 해 지고, 원하는 레벨의 로그메시지들만이 SD 카드로 저장되고 LogCat 을 통해서 출력되도록 할 수 있다.

나. Log 파일 추출 및 서버 전송

LogCat 은 파일로 저장되지 않아 이클립스에서만 로그를 볼 수 있기 때문에 효율적인 로그 관리가 어렵다. 물론 LogCat 을 파일로 저장하는 방법이 존재한다.



[그림 45. LogCat 을 통한 로그파일 저장 예]

[그림 46. LogCat 을 통해 생성된 로그파일]

위 그림과 같이 이클립스 DDMS 의 LogCat 창을 통해서 해당 로그들을 선택하고 저장하기 아이콘을 누르면 원하는 위치에 저장할수 있다. 또는 콘솔창을 이용해 텍스트 파일로 저장하는 코드를 직접 입력하는 것도 가능하다. 그러나 이 방법은 코드를 직접 만든 이클립스가 아닌 콘솔창에 Log Cat 을 저장하는 코드를 매번 직접 입력해야 하는 번거로움이 있다. 그렇기 때문에 배포된 안드로이드 어플리케이션에 심각한 오류가 발생했을 경우, LogCat 을 확인하기 위해서 고객의 폰을 직접 확인해야 하는 불편함이 있다..

그리고 microlog4android 라이브러리를 이용할 경우 SD 카드에 로깅파일들이 저장할수 있지만, 이를 원격 서버로 전송하는 기능은 지원하고 있지 않기 때문에, 로깅 파일 자체를 써드파티 서버로 전송해야 하는 불편함이 있다.
본 logdog 의 실제 적용 사례(Google Play 에서 배포 중인 '알람몬')는 별첨 3 에서 자세히 설명 하였다.

2-5. Sensing-funf

2-5-1. 개요 및 배경

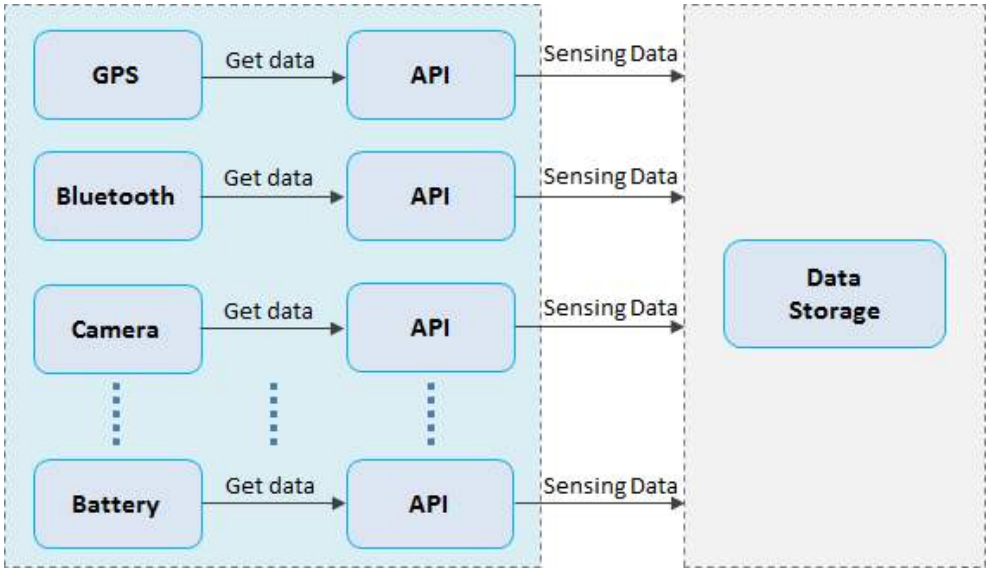
최근 라이프로그에 관련된 기술에 대한 관심이 높아지며 스마트폰의 센서를 활용하여 수집된 데이터에 대한 관심도가 높아지고 있다. 라이프로그(Life Log)란 디지털 장치를 활용하여 일상생활에서 경험하는 모든 정보를 기록 • 검색할 수 있는 기술이다.

개인이 전화한 통화기록, 어디에 갔는지에 대한 정보, 무엇을 먹었는지 등을 디지털 데이터베이스로 만들어 특정 유저가 무엇을 먹었는지, 누구에게 전화를 했는지 등을 기록해 두었다가 해당 유저의 특성에 맞게 적합한 광고를 게재하거나 상품을 추천해 주는 행동 타케팅 광고방식 등이 라이프로그를 활용할 수 있는 서비스다.

예를 들어 휴대폰의 GPS 기능과 연계하면 해당 유저의 그날의 행동반경이나 방문장소 등을 분석하여 유저가 필요로 하는 쿠폰을 보내주는 등의 방법으로 마케팅 활동을 할 수 있을 것이다. 또한 IPTV 에서의 차별화된 광고방식으로도 활용할 수 있다.

이처럼 라이프로그와 이와 연계한 행동 타게팅 광고는 웹의 관심 대상이 군중에서 개인으로 옮겨가고 있는 가운데 가장 각광을 받게 될 것으로 기대되는 분야이다.

모바일시장에서도 현재 다양한 데이터들을 활용한 모바일 앱들을 개발하는 추세이다. 실제로 모바일 기기 내부에는 다양한 센서들이 있다. 그런데 각각의 센서들은 서로 이질적인 인터페이스를 제공하기 때문에, 센서들의 정보를 가져오기 위해선 이에 대한 학습 노력이 선행되어야 한다.



[그림. 47. 각각의 센서에서의 데이터 수집 아키텍쳐]

위의 구조처럼 각각의 센서에서 데이터를 가져온다면 매번 각각의 센서들의 다른 방식의 API 를 활용하여 데이터 요청을 해야한다. 다양한 센서를 활용할 수록 자연적으로 코드양도 증가할 수 밖에 없다. 이로 인한 낮은 개발 생산성은 물론 코드 복잡도도 증가하게 되어서 유지 보수과정에서도 문제가 발생하거나 공수가 증가할 가능성이 매우 높다.

2-5-2. 설계 개선 목표

위에서 언급한 문제점을 해결 하기 위하여 두 가지 목표를 가지고 오픈소스를 조사 하기 시작하였다.

- 다양한 센서들의 데이터를 수집하기 위한 방법에 대하여 일관성 있는 인터페이스를 제공하여 개발자가 각각의 센서에 대한 다른 인터페이스를 잘 몰라도 쉽게 센서들의 정보를 수집하여 적은 학습 곡선을 제공
- 다양한 센서들의 세부 설정을 각각의 센서에서 하는 것이 아니고 모든 센서들을 하나의 설정 컴포넌트에서 제어할 수 있도록 하여 생산성 향상

2-5-3. 아키텍쳐 개선을 위한 사전 활동

2-5-3-1. 아키텍쳐 개선의 시작

본 장에서는 많은 센서들의 수집을 위한 아키텍쳐 개선 작업을 위하여 기존 문제점들을 인식하여 개선하고자 노력한 오픈 소스를 적극적으로 활용하고자 한다.

2-5-3-2. 활용할 오픈 소스 선택

가. Funf

우리는 조사 끝에 Funf 라는 오픈 소스 프레임워크를 적극적으로 활용하기로 하였다.
Funf 는 MIT 미디어 랩에서 개발한 안드로이드용 오픈소스 센싱 프레임 워크로써, 안드로이드 기반의 모바일 디바이스가 가진 GPS, 블루투스, 무선랜, 가속계, 기울기 등 30 여개의 센서가 얻어내는 데이터를 지속적으로 기록하고 전달해준다.

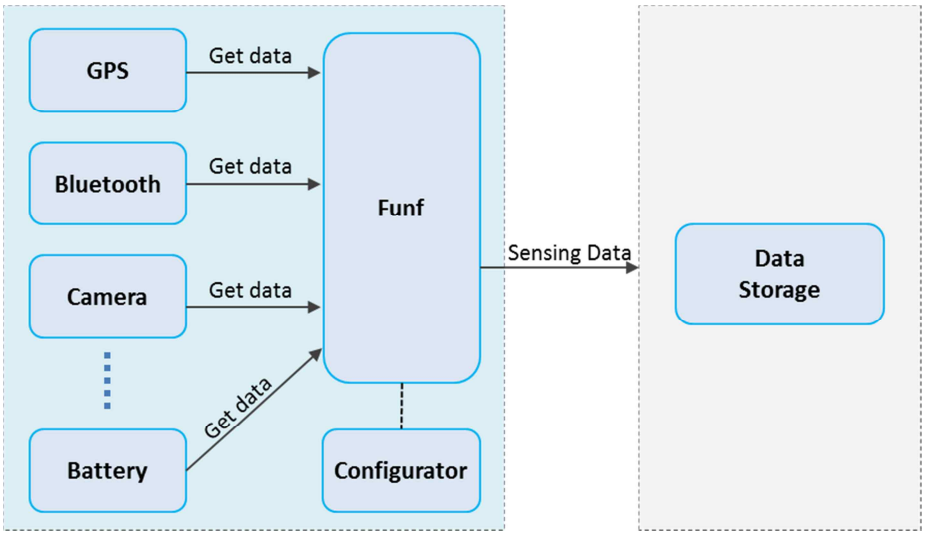
[그림 48. Funf 소개 화면]

주요 기능은 다음과 같다

- 원격 구성.
- 내장 데이터 "Probe"
- 자동 또는 수동 데이터 업로드가 가능
- 서버 연결이 복원 될 때까지 인터넷 연결을 사용할 수 없는 경우는 로컬 데이터를 저장
- 민감한 데이터에 대한 암호화: 텍스트 데이터 또는 전화 번호는 사람이 읽을 수 있는 형식으로 저장하지 않음.
- 로컬에 저장된 데이터베이스 파일의 암호화.
- 수동 데이터 수집을 위한 기본 설문 조사 시스템이다.
- 배터리 소모를 줄여 일상적으로 사용 - 다양한 최적화

Funf 는 안드로이드 기기에 쓰이는 거의 모든 센서들의 정보를 모아서 개발자들에게 제공을 해준다. 그리고 Funf 내부의 구조까지 파악 할 필요없이 3rd-party 개발자 API 를 통해 Funf 프레임 워크의 기능과 서비스를 제공한다.

Funf 를 적용할 시에도 각종 센서들의 제어를 모아서 담당하는 설정 컴포넌트가 존재 하기 때문에 높은 생산성을 보이고 있다.



[그림 49. Funf 의 데이터 수집 아키텍쳐]

나. Funf Probe

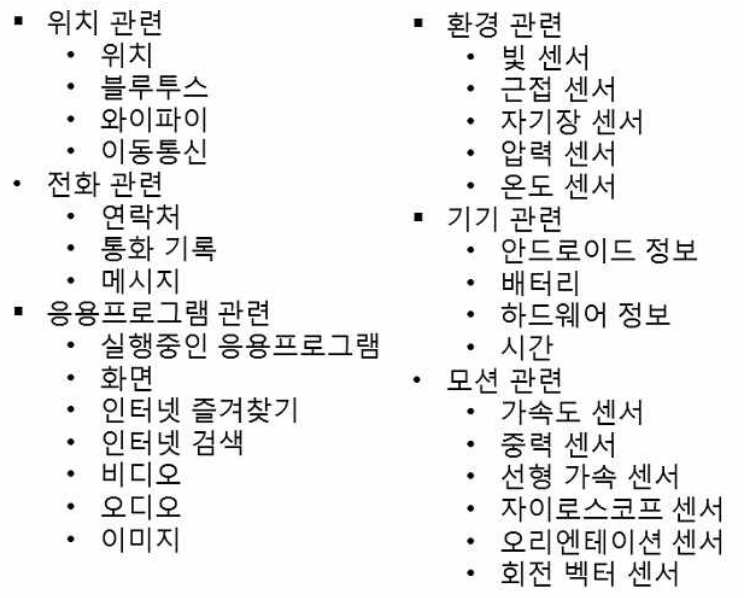
Probe 는 Funf 프레임 워크에서 사용하는 기본 데이터 수집 객체이다. Probe 들은 각각 정해진 특정 유형의 정보를 수집한다. 이렇게 Probe 를 통하여 스마트 폰에서 가속도계, GPS 위치 검사뿐만이 아니라 통화목록, 어플리케이션의 사용목록, 스마트 폰에 저장된 미디어 파일에 대한 정보 등 다양한 데이터를 수집할 수가 있다. 각 Probe 는 원격으로 활성, 비활성화를 조절할 수 있다.

Funf 는 내장 된 Probe 들 말고도 3rd party 개발자가 새로운 Probe 를 추가 할 수 있는 유연한 아키텍처를 가지고 있다. 예를 들어, 사용자가 어느 엑티비티에서 오랫동안 머물러 있는지 등의 유용한 정보를 파악함으로써, 어플리케이션을 개발하고, 디버깅 하는데 도움이 되도록 개발하고 싶다면 개발자는 사용자의 로그가 기록된 데이터를 저장소에 저장 한 다음 서버로 다시 전송한다.

사용자의 개인 정보보호, 배터리 소모, 저장 용량의 부족 같은 문제 등에 대응하는 자신의 코드 모듈을 개발해야 할 것이다. 하지만 Funf 를 활용하면 개발자가 필요로 하는 모든 정보를 기록하고, 이 모든 문제를 처리에 Funf 프레임 워크의 기존 모듈과 경험을

활용하여 간단한 Funf Probe 를 추가 할 수 있다. 또한, 이미 분석을 개선하여 수집 한 다른 센서를 활용하여 개발을 조금 더 편리하게 할 수 있을 것이다.

내장 된 Probe 에 저장되는 전화 번호 및 문자 메시지 같은 민감한 데이터는 메모리에만 가지고 있으며, 일반 텍스트로 저장 하지 않아 개인 정보 보호를 철저하게 지키고 있다.

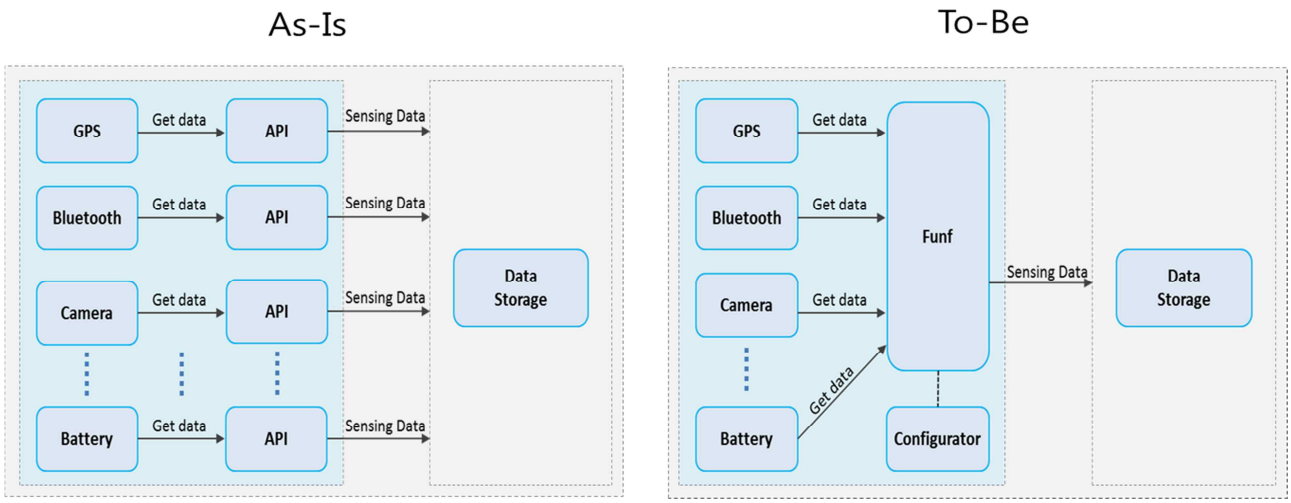


[그림 50. Funf 에서 수집가능한 데이터]

모든 Probe 는 내부의 데이터가 JSON 형식으로 저장이 되어 있다. 각 Probe 에 대한 상세한 정보를 알고 싶다면 Funf Google Code(http://code.google.com/p/funf-open-sensing-framework/wiki/BuiltinProbes)에서 알 수 있다.

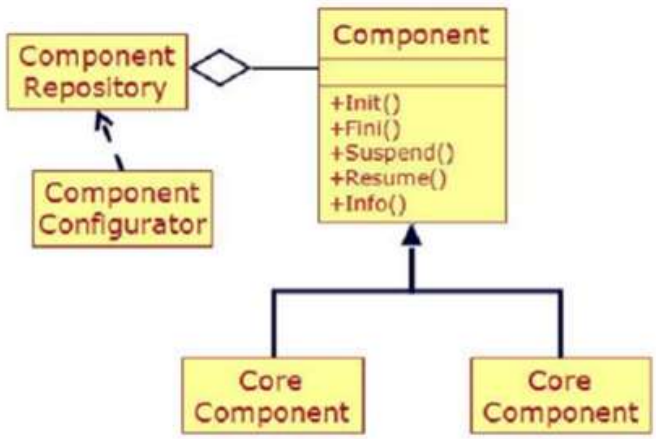
2-5-4. 아키텍쳐 개선 활동

Funf 프레임워크를 사용함으로써 기존(As-Is)에서처럼 많은 센서들의 데이터를 하나하나 따로 수집할필요 없이 사용한 후(To-Be)처럼 Funf 를 통하여 데이터를 모아서 한번에 수집을 할 수가 있으며 설정 컴포넌트를 통하여 각 센서들의 제어를 한번에 할 수 있게 되었다.



[그림 51. sesror 컴포넌트의 현재 구조와 예상 구조]

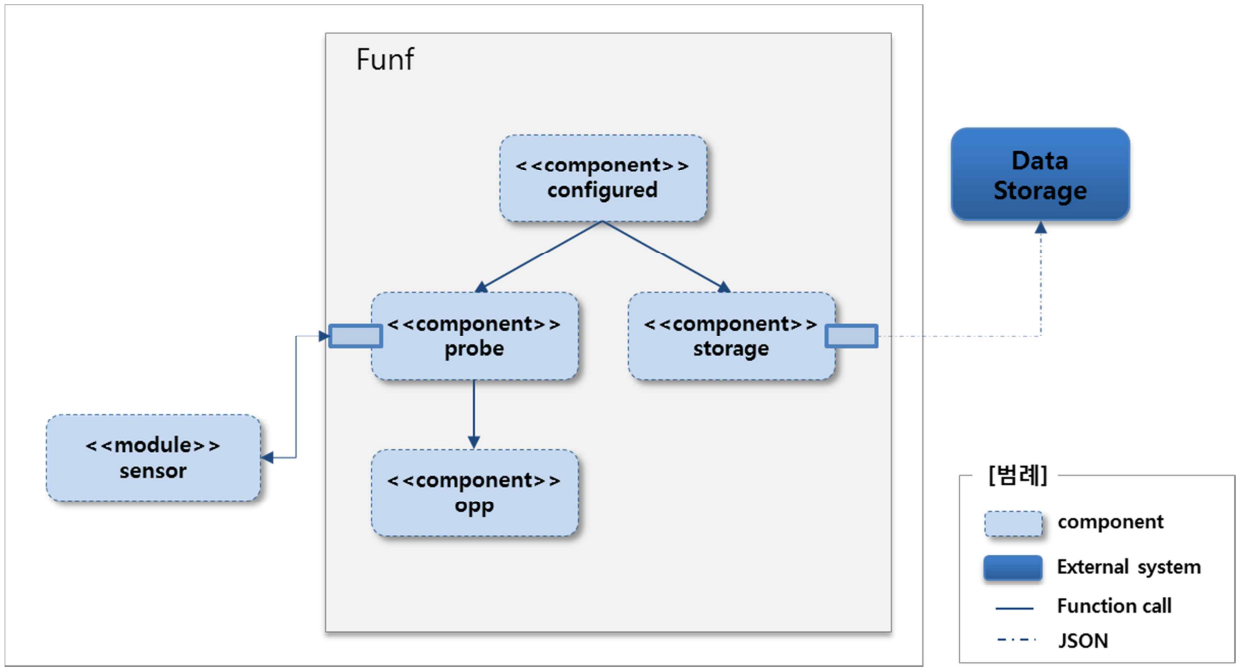
Funf 에서는 Component Configurator 패턴을 활용하였다. Component Configurator 패턴은 런타임 시 사용하는 컴포넌트들을 제어하고, 시스템 중지 없이 컴포넌트를 추가, 제거, 교체할 수 있는 패턴이다.



[그림 52. Component Configurator 패턴]

개발자의 생산성을 높이기 위해서 컴포넌트의 변화가 생기더라도 전체적인 시스템은 문제없이 작동되도록 구현되어 있다.

2-5-5. 아키텍쳐 개선 결과



[그림 53. 개선된 C&C View]

기존에는 각 센서들이 일관성이 없는 인터페이스를 가지고 있어 센서들을 사용하기 위해서 많은 학습이 필요하였다. 게다가 많은 센서들의 데이터를 하나하나 수집해야 하며, 각 센서들의 설정들도 각자 해줘어야 했기 때문에 생산성에도 문제가 있었다.

이러한 문제를 해결하기 위해, 기존 오픈소스 Funf 를 활용하여 다음과 같이 컴포넌트들을 구성하였다.

- configured: 각 센서들의 세부 설정에 따른 제어 컴포넌트
- opp: Object Push Profile 블루투스 통신 프로파일
- storage: database 관련 코드
- probe: Sensor 정보 가져오기-service

그에 따라, “Wifi”센서를 제어하는 설정 컴포넌트로 봤을 때 개발 생산성 개선, 관리 용이성 개선들이 이루어 졌다.

```
{
    "name": "WifiScanner",
    "version":1,
    "dataArchivePeriod":3600,
    "dataRequests":{
```

```
            "edu.mit.media.funf.probe.builtin.LocationProbe": [
                { "PERIOD": 900, "DURATION": 30 }
            ],
            "edu.mit.media.funf.probe.builtin.WifiProbe": [
                { "PERIOD": 900 }
            ]
        }
    }
```

[그림 54. 설정 컴포넌트 예제]

자세한 설명 및 예제는 별첨 4 를 통해서 설명을 하였다.

2-6. Android Billing Lib

2-6-1. 개요 및 배경

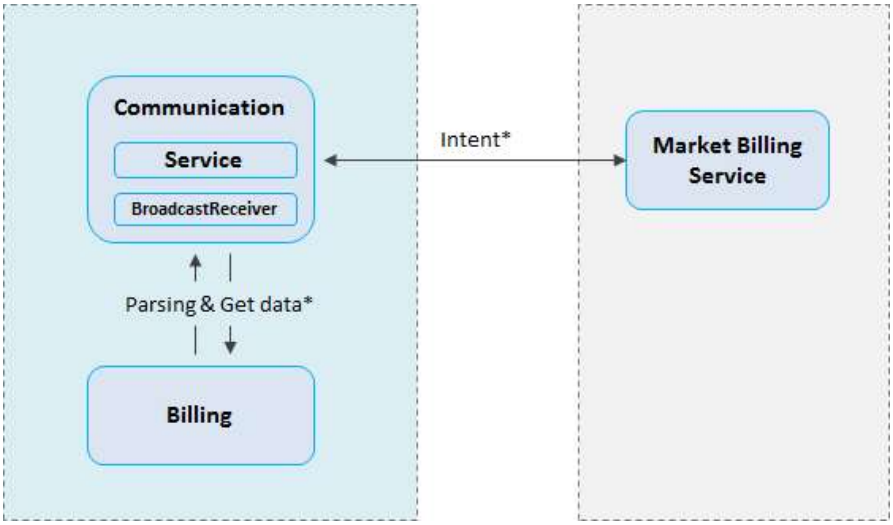
모바일 앱 시장에서 FREEmium 경제의 효과가 증명되고 있다. 비즈니스인사이더의 최근 조사 결과 가장 높은 수익률을 자랑하는 앱은 대부분 무료앱으로서, 아이폰의 경우 최고 수익앱의 2/3 이 무료 다운로드를 제공하는 앱들이다. 이런 무료 앱들은 광고보다는 무료 다운로드를 통한 in-app purchase 를 유도하는 수익전략을 추구하여, 매출의 100%를 달성하고 있다. 징가, 판도라, 블리츠, 배트 등이 이런 수익 전략을 통해 성공한 사례들이며, 최근 국내에서는 애니팡, 드래곤 플라이트등이 이러한 모델에 해당된다. 이러한 FREEmium 모델이 IOS 에서는 성공적으로 안착하고 있지만, 안드로이드에서는 아직 IOS 에 비해서 구현이 복잡하여 활성화가 덜 되어 있다.

[그림 55. 드래곤 플라이트]

[그림 56. 애니팡]

사용자가 구글 플레이에서 다운받은 어플리케이션 내에 추가적인 유료 컨텐트나 서비스 또는 기능을 제공하고자 하는 개발자는 그 결제 수단으로 반드시 구글 플레이의 in-app purchase 결제 시스템을 사용해야 한다. 즉, 구글 플레이에서 유료 앱 또는 유료 상품을 구매하면 구글 플레이에서 자동으로 통신사와 연결하거나 신용카드 정보 등을 이용해서 터치 몇번으로 결제를 상당히 편리하게 할수있다.

안드로이드 플랫폼을 활용한 모바일 소프트웨어 개발 시, in-app purchase 를 구현하기 위해서는 MarketBillingService 와 많은 메시지를 주고받고, json 데이터를 사용해야 하는데 이 과정에서 프로토콜에 맞춰 복잡한 json 데이터를 생성하고 메시지를 보내야 하는 등 많은 개발 공수가 든다.



[그림 57. Billing 서비스를 위한 파싱]

in-app purchase 구현을 위한 메시지 흐름과 json 파싱은 위의 그림과 같은 구조로 이루어지는데, 안드로이드 어플리케이션 개발자가 in-app purchase 를 구현하기 위해서는 market Billing Service 와 안드로이드 앱과의 흐름을 제어해야 하고, 거래 정보에 대한 복원, 서드파티 서버와의 동기화, 환불 등을 고려해야 하기 때문에 개발하기에 많은 어려움이 있을수 있고 보안과 관련된 부분에서도 문제가 발생할 가능성이 높기 때문에 in-app purchase 를 구현하기 위해서는 반복적이며, 많은 학습 노력이 선행되어야 한다.

가. 안드로이드 마켓 In-app purchase

1) In-app Billing Service

In-app Billing Service 는 in-app purchases 를 처리하는 Google Play 서비스이다. 안드로이드 앱은 특정 인앱제품에 대한 구매(checkout) 요청을 In-app Billing Service 에 보내게 되고. In-app billing Service 는 요청 및 지불 양식을 검증하고 금융 거래를 포함하여 결제 정보를 모두 처리한다. 체크아웃 과정이 완료되면, 이 서비스는 안드로이드 앱에 주문번호, 주문 시간, 지불된 가격과 같은 구매정보를 전송한다.

2) Product Type

In-app Billing 은 In-app products 와 Subscriptions 두가지 타입의 Product 를 지원한다. In-app products 은 사용자가 한번만 구입할 때 사용되는 항목이다. 예를 들어, 디지털 컨텐츠를 구매하거나, 일회성 요금을 지불하거나, 앱의 기능을 잠금해제하 할 때 사용된다. In-app products 는 환불창이 없기 때문에, 환불을 하기 위해서는 판매자가 직접 환불을 처리해야한다.

Subscriptions 은 사용자가 subscription 을 구입하면, 구글 플레이의 결제 프로세서가 자동으로 사용자의 계정에 지정된 시간 간격으로 청구를 한다. 사용자는 언제든지 구독권을 취소할수 있지만, 환불창은 따로 존재하지 않기 때문에 직접 개발자에게 문의해야 한다.

[그림 58. 환불하기]

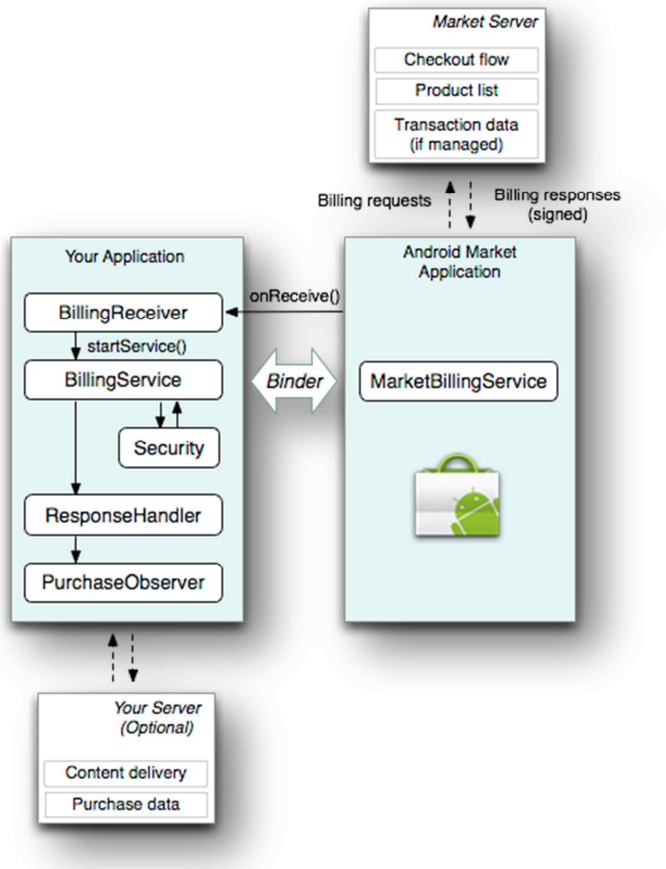
3) Purchase Type

In-app Billing Service 는 “Managed per user accout” 와 “Unmanaged’ 두가지 구매유형을 제공한다. 구글 플레이에서 사용자가 "managed per user account" 구매 유형을 사용하여 아이템을 구입할 때, 구글 플레이는 영구적으로 사용자 별 기준으로 각 상품에 대한 거래 정보를 저장한다. 이것은 나중에 특정 유저가 해당 아이템을 구입한 상태인지를 여부를 구글플레이 서비스를 통해서 얻을수있다. 만약 사용자가 이미 구입한 상품을 다시 구입하려고 시도하면 구글 플레이는 해당 상품을 다시 구입하는 오류를 방지하고 "Item already purchased" 으로 표시된다. 개발자가 중복구매 할 필요가 없는 아이템을 판매하고자 한다면, "managed per user account" 구입 유형이 유용하다. 이는 사용자가 응용프로그램을 지웠다가 다시 설치했을 때 또는 새로운 기기에서 다시 설치시에도 복원 가능한 장점이 있다.

사용자가 “ Unmanaged” 유형의 상품을 구매한다면, 구글 플레이에 거래 정보가 저장되지 않는다. 이것은 이러한 아이템들에 대한 거래정보를 구글플레이에서 검색할수 없다는 것을 의미한다. 즉, 개발자가 거래정보 관리에 대한 책임을 져야한다. "unmanaged" 유형은 여러 번 판매할수 있는 게임의 마법 주문이나 연료와 같은 소모품 아이템을 판매하고자 할 때 유용하다.

4) In-app Billing Architecture

Google Play App 은 안드로이드 앱과 구글 플레이 서버간의 결제(billing) 요청과 응답을 전달하기 위해서, 비동기 메시지 루프를 사용한다. 안드로이드 앱은 구글 플레이 서버와 직접 통신해서는 안된다. 대신, 안드로이드 앱은 결제 요청을 IPC(Interprocess Communication)를 통해서 Google Play App 에 하고, 구매여부를 구글 플레이 어플리케이션으로부터 비동기 브로드캐스트 인텐트 형식으로 응답받는다. 인앱 결제시 안드로이드 앱에서 모든 보안 관련 작업을 처리할수도 있지만, 원격 서버에서 보안관련 작업을 수행하는 것이 보안 공격의 취약성을 더 줄일수 있다.



[그림 59. In-app Billing Architecture]

위 그림을 보면 안드로이드 앱이 인앱 결제를 수행할 때 Android Market Application 을 통해서 결제 메시지들을 송수신하고, Market Server 와는 직접적으로 통신하고 있지 않음을 확인할수 있다. 여기서 BillingService 는 안드로이드 앱의 구매 메시지들을 처리하고, 결제 요청을 In-app Billing Service 에 요청한다. BillingReceiver 는 Android Market Application 으로부터 모든 비동기식 결제 응답을 수신한다. Security(보안 컴포넌트)는 서명확인, nonce 생성과 같은 보안 관련 작업들을 수행한다. ResponseHandler 는 구매

알림, 에러, 다른 상태 메시지들의 안드로이드 앱에서의 특정 과정을 제공한다. PurchaseObserver 는 안드로이드 앱에 구매 정보와 상태정보를 업데이트 하라는 콜백을 보낸다.
이러한 컴포넌트들 뿐만 아니라 안드로이드 앱은 사용자가 아이템을 구매할수 있도록 UI 를 제공해야 하고, 사용자의 구매 기록에 대한 정보를 저장해야한다. 사용자가 인앱 결제를 시작할 때, Google Play Application 의 체크아웃 UI 가 나타나게 되며, 결제가 완료되면, 안드로이드 앱이 다시 시작된다.

5) In-app Billing messages

사용자가 인앱 구매를 요청하면, 안드로이드 앱은 IPC 함수를 호출하여 MarketBillingService 에 인앱 결제 메시지들(In-app Billing messages)을 전송한다. Android Market Application 은 안드로이드 앱에 상태 알림 및 다른 정보들을 제공하는데, 모든 결제 요청에 대해 동기적으로 응답한다. Google Play Application 은 에러 메시지들과 상세 결제관련 정보들을 제공하는데, 몇몇 결제 요청에 대해서 비동기적으로 응답한다.
안드로이드 앱은 MarketBillingService 인터페이스에 의해 토출된 sendBillingRequest() IPC 메소드를 호출하여 인앱결제 요청을 보낸다. 이 인터페이스는 AIDL(Android Interface Definition Language)파일인 IMarketBillingService.aidl 이라 정의된다. sendBillingRequest() 메소드는 단일 Bundle 매개변수를 가지고 있다. Bundle 에 인앱 요청을 위해 다양한 매개변수들의 여러 키 값들의 쌍을 반드시 포함해서 보내야 한다. 인앱 요청중 가장 중요한 Bundle 의 키는 “BILLING_REQUEST”이다. 이 키는 결제요청의 유형을 지정한다. 다음은 지원되는 다섯가지 결제 요청 타입이다.

- CHECK_BILLING_SUPPORTED

이 요청은 사용자의 Google Play Application 이 In-app billing 을 지원하는지 확인한다.

- REQUEST_PURCHASE

이 요청이 보내지면, Google Play Application 에 구입 메시지를 보낸다. 그러면 Google Play Application 은 결제(checkout) UI 를 보여주게 된다.

- REQUEST_PURCHASE

이 요청은 구매 상태 변화에 대한 상세 내용들을 검색한다. 구매 상태는 구매가 성공적으로 결제되었거나, 구매가 취소되었거나, 이전에 구매했던 상품이 환불되었을 때 변경되는데, 이런 경우 Google Play Application 은 안드로이드 앱에 구매상태가 변경되었음을 알려주며, 이때 이 요청을 통해서 구매 상태 변화에 대한 상세 내용들을 확인할수 있다.

▪ CONFIRM_NOTIFICATIONS
이 요청은 안드로이드 앱이 구매 상태의 변경사항을 받았다는 것을 Google Play Application 에 알려준다. 만약 이 요청을 보내지 않는다면, google Play Application 은 변경된 구매상태 변경 알림을 게속 보내게 된다.

▪ RESTORE_TRANSACTIONS
이 요청은 안드로이드 앱 사용자의 결제 내역을 검색한다.

6) In-app Billing Responses
Google Play Application 은 인앱 결제 요청에 대해서 동기식, 비동기식 두가지로 응답한다. 동기식 응답은 다음 세가지 키를 가진 Bundle 을 응답한다.

▪ RESPONSE_CODE
이 키는 상태 정보 및 요청에 대한 오류 정보를 제공한다.

▪ PURCHASE_INTENT
이 키는 checkout UI 를 실행시키는데 사용되는 PendingIntent 를 제공한다.

▪ REQUEST_ID
이 키는 안드로이드 앱의 요청과 비동기식 응답을 매칭하는데 사용할수 있는 요청 식별자를 제공한다. 이러한 키들중 일부는 모든 요청들과 관련이 없을수 있다. 비동기식 응답 메시지는 다음을 포함하여 개별 Broadcast Intent 의 형태로 전송된다.

▪ com.android.vending.billing.RESPONSE_CODE
이 응답 메시지는 개발자가 결제 요청을 한 이후에 전송되며, 결제 요청이 성공적으로 보내졌는지, 또는 결제 요청중에 에러가 발생되었는지 등을 나타내는 Google play server 의 응답 코드가 포함되어있다.

▪ com.android.vending.billing.IN_APP_NOTIFY
이 응답 메시지는 구매가 성공, 취소 또는 환불되었을 때 구매상태가 변화되었다고 알려주기 위해서 전송되며, 하나 이상의 알림 ID 를 포함한다. 각 알림 ID 는 특정 서버측 메시지와 부합되며, 각 메시지들은 하나 이상의 거래에 대한 정보를 포함한다.

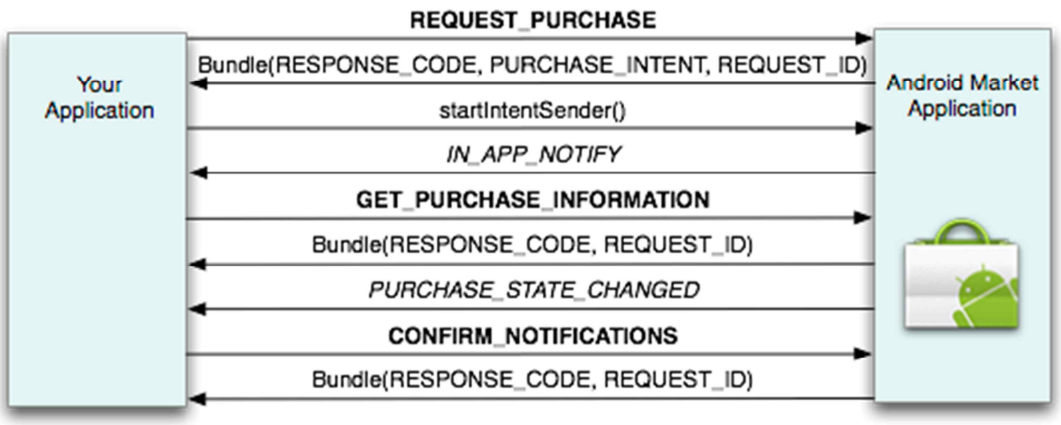
▪ com.android.vending.billing.PURCHASE_STATE_CHANGED

이 응답 메시지는 하나 이상의 거래에 관한 상세 정보를 JOSN 문자열로 포함하고 있다. 이 JSON 문자열은 서명되어있고, 서명도 함께 포함되어있기 때문에, 이 JSON 문자열의 서명을 검증할수 있다. JSON 문자열의 예는 다음 그림과 같다.

```
{ "nonce" : 1836535032137741465,
  "orders" :
    [{ "notificationId" : "android.test.purchased",
       "orderId" : "transactionId.android.test.purchased",
       "packageName" : "com.example.dungeons",
       "productId" : "android.test.purchased",
       "developerPayload" : "bGoa+V7g/yqDXvKRqq+JTFn4uQZbPiQJo4pf9RzJ",
       "purchaseTime" : 1290114783411,
       "purchaseState" : 0,
       "purchaseToken" : "rojeslcdyyiapnqcynkjyyjh" }]
}
```

[그림 PURCHASE_STATE_CHANGED 에 포함된 JSON 문자열의 예]

7) Messaging sequence

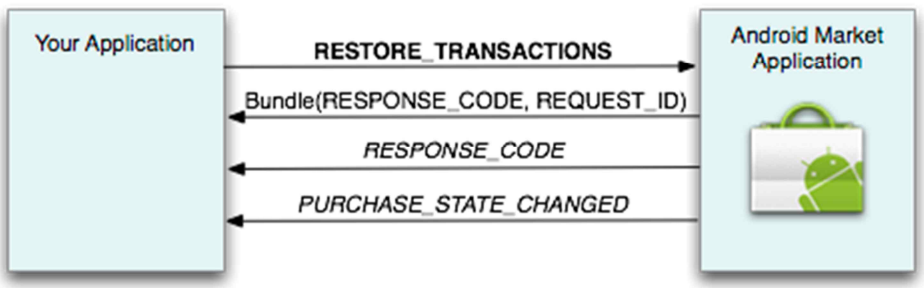


[그림 60. 구매 요청에 대한 메시지 흐름]

일반적인 구매 요청에 대한 메시지 흐름은 위의 그림과 같다. 각각의 sendBillingRequest() 메서드에 대한 요청 유형은 볼드체로, broadcast Intent 들은 이탤릭채로 보인다. 일반적인 구매 요청의 메시지 흐름은 다음과 같다.

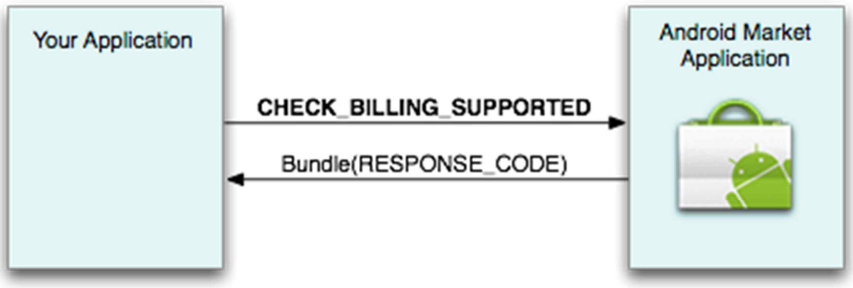
① 안드로이드 앱은 Android Market Application 에 특정 상품의 ID 와 그밖의 매개변수 정보들이 포함된 REQUEST_PURCHASE 요청을 한다.

② Google Play application 은 REQUEST_PURCHASE 요청을 받은 뒤에 RESPONSE_CODE, PURCHASE_INTENT, 그리고 REQUEST_ID 키와 대응되는 값들이 담긴 Bundle 객체를 안드로이드 앱에 동기적으로 보낸다.
③ 안드로이드 앱은 PURCHASE_INTENT 키가 제공하는 PendingIntent 를 실행하면, checkout UI 가 실행된다. PendingIntent 를 실행할 때 반드시 어플리케이션 Context 가 아닌 액티비티 Context 로부터 실행해야 한다.
④ checkout UI 에서 사용자가 상품을 구입했거나 취소했을 경우, Android Market Application 은 안드로이드 앱에 거래 정보를 참조할수 있는 notification ID 를 포함한 IN_APP_NOTIFY broadcast intent 를 보낸다.
⑤ 안드로이드 앱은 notification ID 에 대한 상세한 거래 정보를 얻기 위해서 GET_PURCHASE_STATE_CHANGED 요청을 보낸다.
⑥ Google Play application 은 GET_PURCHASE_STATE_CHANGED 요청을 받은뒤에 RESPONSE_CODE 와 REQUEST_ID 키와 대응되는 값들이 담긴 Bundle 객체를 안드로이드 앱에 동기적으로 보낸다.
⑦ Google Play application 은 notification ID 에 대한 거래정보를 PURCHASE_STATE_CHANGED broadcast Intent 에 담아서 안드로이드 앱에 비동기적으로 보낸다.
⑧ 안드로이드 앱은 REQUEST_ID 를 통해 어떤 요청에 대한 거래정보인지를 확인하고, Google Play application 에 CONFIRM_NOTIFICATIONS 를 보내서 해당 notification ID 에 대한 거래정보를 받았는지를 알려준다. 이 과정을 수행하지 않으면 Google Play application 은 IN_APP_NOTIFY broadcast Intent 를 계속 재전송한다.
⑨ Google Play application 은 CONFIRM_NOTIFICATIONS 요청을 받은 뒤에 RESPONSE_CODE 와 REQUEST_ID 키와 대응되는 값들이 담긴 Bundle 객체를 안드로이드 앱에 동기적으로 보낸다.



[그림 61. 복원과정을 위한 메시지 흐름]

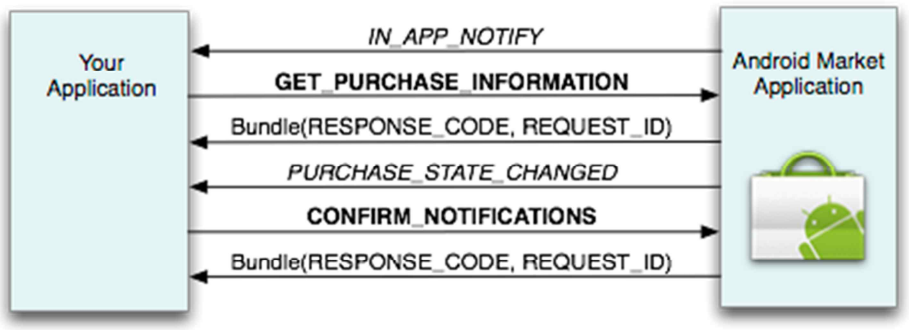
사용자가 안드로이드 앱이 재설치 되었거나, 다른 디바이스에서 처음 설치되었을 경우, 상품 구매 내역에 대해서 복원해줄 필요가 있다. 복원 과정을 위한 메시지 흐름은 위의 그림과 같다. 각각의 sendBillingRequest() 메서드에 대한 요청 유형은 볼드체로, broadcast Intent 들은 이탤릭채로 보인다. 안드로이드 앱이 Android Market Application 에 RESTORE_TRANSCTION 요청을 보내게 되면, 동기적으로 RESPONSE_CODE 와 REQUEST_ID 키와 대응되는 값들이 담긴 Bundle 객체를 동기적으로 받게 된다. 그리고 Android Market Application 은 비동기적으로 요청에 대한 상태 정보나 오류 정보가 포함된 RESPONSE_CODE broadcast Intent 를 전송한다. 안드로이드 앱은 REQUEST_ID 를 통해서 RESTORE_TRANSACTIONS 에 대한 응답임을 확인할수 있다. 또한 Android Market Application 은 거래정보가 담긴 PURCHASE_STATE_CHANGED broadcast Intent 을 비동기적으로 보낸다.



[그림 62. In-app Billing 이 지원되는지 여부를 확인하는 메시지 흐름]

위 그림은 In-app Billing 이 지원되는지 여부를 확인하는 메시지 흐름을 보인다. CHECK_BILLING_SUPPORTED 요청에 대한 동기적 응답으로 In-app Billing 이 지원되는지 여부를 알려주는 RESPONSE_CODE 응답코드가 담긴 Bundle 을 제공한다. RESPONSE_CODE 키에 대한 값이 RESULT_OK 라면 In-app Billing 가 지원되는것이며, RESULT_BILLING_UNAVAILABLE 라면 In-app Billing 빌링이 지원되는 않음을 뜻한다.

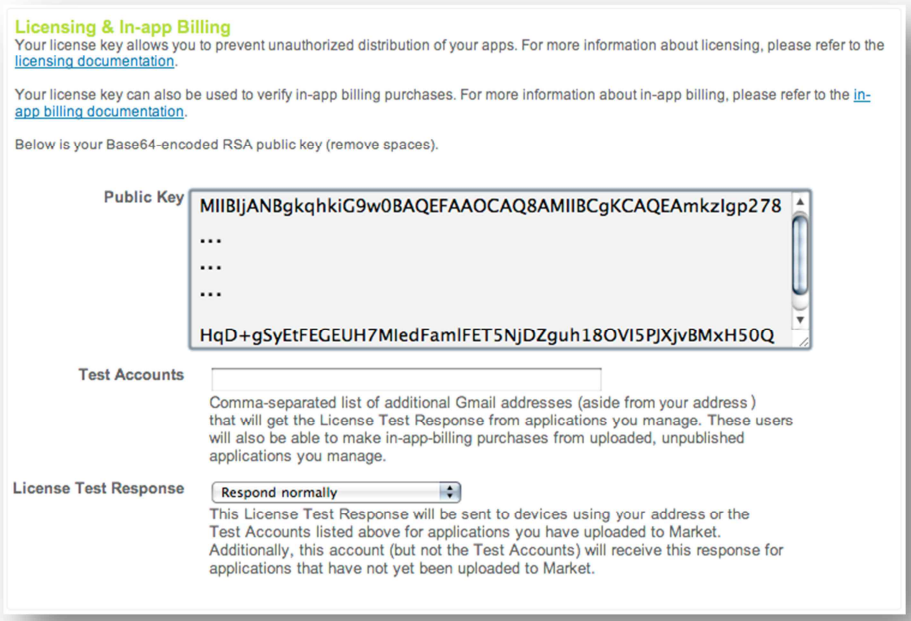
안드로이드 앱이 REQUEST_PURCHASE 메시지를 보내지 않았다 하더라도 Android Market Application 이 IN_APP_NOTIFY broadcast intent 를 보내는 경우가 있다. 다음 그림은 이에 대한 메시지 흐름을 보인다.



[그림 63. IN_APP_NOTIFICATION 에 대한 메시지 흐름]

첫번째 경우는 한 사용자가 두개 이상의 디바이스에 동일한 안드로이드 앱을 설치하고 한 개의 단말에서 in-app purchase 를 시도하였을 때 다른 디바이스의 안드로이드 앱에서도 IN_APP_NOTIFICATION 메시지를 받을수 있다. 이는 “managed per user account” 구매유형의 상품에만 적용된다.
두번째 경우는 Google checkout 판매계정을 통해서 상품을 환불 하였을 경우 Android Market Application 은 환불 알림을 받게 되고, 안드로이드 앱에 IN_APP_NOTIFITY 메시지를 전송한다.

8) Security Controls
Android Market Application 이 거래정보를 안드로이드 앱에 보낼 때 거래정보의 무결성을 보장하기 위해서, PURCHASE_STATE_CHANGED broadcast Intent 에 포함된 JSON 문자열에 서명을 한다. Android Market Application 은 서명을 만들 때 판매자 계정과 연결된 private key 를 사용한다. 그리고 다음 그림과 같이 판매자 계정의 프로필 페이지에서 해당 키 쌍의 public key 부분을 찾을수 있다.
Android Market Application 이 결제 응답을 서명할 때, 서명된 JSON 문자열(암호화 되지 않음)과 서명(SIGNATURE)이 포함된다. 안드로이드 앱이 서명된 응답을 받았을 때 서명 확인을 위해서 판매자 계정의 public key 를 사용한다. 이를 통해서 응답들이 위조되었는지 등에 대한 검증을 수행할수 있다.
또한 in-app Billing Service 는 Android Market Application 으로부터 반환된 구매 정보의 무결성 검증을 위해서 nonce(일회용 무작위수)를 사용한다. 안드로이드 앱은 nonce 를 생성한뒤 GET_PURCHASE_INFORMATION 과 RESTORE_TRANSACTIONS 요청을 할 때 함께 보내야한다. Android Market Application 는 그 요청을 받고, JSON 문자열에 nonce 를 추가한뒤 서명하고 안드로이드 앱에 반환한다. 안드로이드 앱은 JSON 문자열에 대해 서명 확인 뿐만 아니라 nonce 값이 동일한지를 검증해야 한다.



[그림 64. 판매자 계정의 프로필 페이지의 public key]

나. 기존 구현 방법

In-app Billing 구현을 위해서는 Google Play 를 통해서 배포된 Android Application 에서만 구현되어야 하며, 판매자는 Google Wallet Merchant 계정을 가지고 있어야 한다. 다음은 Android SDK 에서 제공하고 있는 In-app Billing 샘플 프로젝트를 통해서 기존 구현 방법을 설명한다.

1) Downloading the Sample Application

In-app billing sample application 은 Google Play 에 in-app billing 요청을 보내고, Google Play 로부터 받은 동기적, 비동기적 응답들에 대해서 무결성을 확인 한 뒤 처리하고, in-app 상품의 결제를 위한 UI 를 제공하고 있다.

[표 in-app billing sample application 소스 파일들]

파일	설명
IMarketBilling Service.aidl	Google Play 의 in-app billing service 에 대한 IPC 인터페이스를 AIDL(Android Interface Definition

	Library) 파일로 정의
Dungeons.java	구입 내역을 보여주거나 구입할수 있도록 UI 를 제공
PurchaseDatabase.java	구입 내역을 저장하기 위한 로컬 데이터베이스
BillingReceiver.java	Google Play 로부터 broadcast intents 를 받은뒤 BillingService 로 전달
BillingService.java	MarketBillingService 에 바인딩해서 Google Play Application 에 메시지를 보내는 서비스
ResponseHandler.java	PurchaseDatabase 와 UI 를 업데이트하는 메서드들이 포함
PurchaseObjserver.java	구매 과정에서 발생되는 변화들을 관찰하는 추상클래스
Security.java	다양한 보안 메서드를 제공
Const.java	다양한 Google Play 상수들을 정의하고, 기타 상수들을 정의
Base64DecoderException.java and Base64.java	2 진수 값을 Base64 인코딩으로 변환하는 기능 제공

2) Running the sample application

In-app billing sample application 이 작동되도록 하기 위해서는 Google Play 판매자 계정의 public key 를 Security 클래스의 String base64EncodePublicKey = "your public key here" 부분에 넣고, Google Play 에서 배포할수 있도록 패키지명을 변경 해야 한다. 그런뒤에 앱의 릴리즈 버전을 빌드하고 서명한뒤 Google Play publisher site 에 게시(publish)되지 않은 상태로 업로드한다. 그런뒤에 판매할 in-app 상품들을 게시한다.

3) Manifest 설정

in-app purchase 를 위해서는 다음과 같이 Google Play 에 요청을 보낼수 있도록 com.android.vending.BILLING 퍼미션을 메니페스트 파일에 추가하고, 서비스를 선언해야

한다. 그리고 Google Play 로부터 broadcast intents 를 수신하기 위해서 BroadcastReceiver 를 선언해야 한다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.dungeons" android:versionCode="1"
android:versionName="1.0">
<uses-permission android:name="com.android.vending.BILLING" />
<application android:icon="@drawable/icon"
android:label="@string/app_name">
<activity android:name=".Dungeons" android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<service android:name="BillingService" />
<receiver android:name="BillingReceiver">
<intent-filter>
<action android:name="com.android.vending.billing.IN_APP_NOTIFY" />
<action android:name="com.android.vending.billing.RESPONSE_CODE" />
<action
android:name="com.android.vending.billing.PURCHASE_STATE_CHANGED" />
</intent-filter>
</receiver>
</application>
</manifest>
```

4) Local Service 생성

안드로이드 앱은 Google Play 와의 메시지를 송수신 하기 위한 로컬 서비스가 필요하다. 이 로컬 서비스는 첫번째로 다음 코드와 같이 MarketBillingService 를 바인딩해야한다.

```
try {
  boolean bindResult = mContext.bindService(
    new Intent("com.android.vending.billing.MarketBillingService.BIND"), this,
    Context.BIND_AUTO_CREATE);
  if (bindResult) {
    Log.i(TAG, "Service bind successful.");
  } else {
    Log.e(TAG, "Could not bind to the MarketBillingService.");
  }
} catch (SecurityException e) {
  Log.e(TAG, "Security exception: " + e);
}
```

서비스를 바인딩 했다면 다음과 같이 IMarketBillingService 인터페이스에 대한 참조를 만들어야 IPC 메소드 호출을 통한 in-app Billing 요청을 할수 있다.

```
/**
 * The Android system calls this when we are connected to the MarketBillingService.
 */
public void onServiceConnected(ComponentName name, IBinder service) {
  Log.i(TAG, "MarketBillingService connected.");
  mService = IMarketBillingService.Stub.asInterface(service);
}
```

5) 결제 요청 보내기

MarketBillingService 인터페이스는 Bundle 매개변수를 요구하는 sendBillingRequest 메소드를 제공한다. Bundle 에 다양한 key-value 쌍을 사용하여, 요청의 유형을 지정할수 있는데 다음 표에서 보이고 있다.

Key	Value Type	Possible Value
BILLING_REQUEST	String	CHECK_BILLING_SUPPORTED, REQUEST_PURCHASE, GET_PURCHASE_INFORMATION,

		CONFIRM_NOTIFICATIONS, RESTORE_TRANSACTIONS
API_VERSION	int	1, 2
PACKAGE_NAME	String	A valid package name
ITEM_ID	String	Any valid product identifier
NONCE	long	Any valid long value
NOTIFY_IDS	Array of long values	Any valid array of long values
DEVELOPER_PLAYLOAD	String	Any valid String less than 256 characters long

[표 sendBillingRequest 요청을 통해서 전달되는 Bundle 키들에 대한 설명]

위 표에서 BILLING_REQUEST 는 결제 요청의 유형을 지정한다. API_VERSION 는 Google Play 의 in-app billing service 의 버전을 뜻하며, 현재는 버전 2 이다. PACKAGE_NAME 은 in-app billing 을 요청하는 안드로이드 앱의 패키지명이다. ITEM_ID 는 in-app 요청을 위해 Google Play publisher site 에서 지정한 상품의 ID 이다. NONCE 는 Google Play 로부터 온 거래정보의 무결성을 확인하기 위해서 사용되는 값이다. NOTIFY_IDS 는 구매 상태가 변경된 상품들의 ID 배열이다. DEVELOPER_PLAYLOAD 는 주문에 대한 추가 정보를 저장하기 위해서 사용된다.

결제 요청을 하기 위해서는 첫번째로 아래 그림과 같이 BILLING_REQUEST, API_VERSION 과 PACKAGE_NAME 키는 반드시 Bundle 에 포함시켜야 한다.

```
protected Bundle makeRequestBundle(String method) {
  Bundle request = new Bundle();
  request.putString(BILLING_REQUEST, method);
  request.putInt(API_VERSION, 1);
  request.putString(PACKAGE_NAME, getPackageName());
  return request;
```

6) in-app billing 이 지원되는지 확인하기

다음 코드는 Google Play Application 이 in-app billing 을 지원하는지, 어떤 API 버전을 지원하는지 확인한다. mService 는 MarketBillingService 인터페이스 객체이다.

```
/**
* Request type is CHECK_BILLING_SUPPORTED
*/
  Bundle request = makeRequestBundle("CHECK_BILLING_SUPPORTED");
  Bundle response = mService.sendBillingRequest(request);
  // Do something with this response.
}
```

CHECK_BILLING_SUPPORTED 요청은 Bundle 객체를 반환하는데 이것은 RESPONSE_CODE 키가 포함되어있다. 그리고 RESPONSE_CODE 키는 다음과 같은 값들을 가질수 있다.

- RESULT_OK : 해당 in-app billing 버전이 지원됨을 의미
- RESULT_BILLING_UNAVAILABLE : in-app billing 이 지원되지 않음을 의미
- RESULT_ERRO : Google Play Application 에 연결 실패
- RESULT_DEVELOPER_ERROR : BILLING 권한이 선언되지 않았거나, 정당한 서명이 아니거나, 잘못된 요청을 했음을 의미

7) 구매 요청

다음 코드는 in-app 상품에 대해 구매 요청하는 법을 보여준다. mProductID 는 Google Play 에서 in-app 상품의 product ID 이다.

```
/**
* Request type is REQUEST_PURCHASE
*/
  Bundle request = makeRequestBundle("REQUEST_PURCHASE");
  request.putString(ITEM_ID, mProductId);
  // Request is for a standard in-app product
  request.putString(ITEM_TYPE, "inapp");
  // Note that the developer payload is optional.
  if (mDeveloperPayload != null) {
    request.putString(DEVELOPER_PAYLOAD, mDeveloperPayload);
  }
  Bundle response = mService.sendBillingRequest(request);
  // Do something with this response.
```

REQUEST_PURCHASE 요청은 Bundle 객체를 반환하는데 이것은 BILLING_REQUEST, API_VERSION 그리고 PACKAGE_NAME 키가 포함되어있다. RESPONSE_CODE 키는 요청의 상태를 제공하며, REQUEST_ID 키는 요청에 대한 고유한 요청 식별자를 제공한다. 그리고 PURCHASE_INTENT 키는 PendingIntent 를 제공하는데, 이는 Google Play 상품 구매 UI 를 실행시킬수 있다.

REQUEST_PURCHASE 요청으로 인해 두 가지 broadcast intents 를 받을수 있다. 첫 번째로 Google Play Application 은 RESPONSE_CODE broadcast intent 를 보낸다. 이는 REQUEST_PURCHASE 요청이 성공적으로 전송되었음을 의미한다. 다음으로 사용자가 구매를 하거나 구매 취소 하였을 경우 Google Play Application 은 IN_APP_NOTIFY broadcast intent 를 보낸다. 이 메시지는 notification ID 를 포함하고 있어 이를 이용해서 REQUEST_PURCHASE 요청에 대한 구매정보 상세 정보를 검색할 수 있다.

8) 구매 또는 환불에 대한 정보 검색하기

다음 코드는 in-app 상품에 대한 구매 정보를 검색하는 방법을 보인다. mNonce 는 구매 정보의 무결성을 확인하기 위해서 생성한 암호화된 값으로, PURCHASE_STATE_CHANGED broadcast intent 와 함께 반환된다. mNotifyIds 는 notification ID 들의 배열이며 IN_APP_NOTIFY broadcast intent 에 포함되어 있다.

```
/**
* Request type is GET_PURCHASE_INFORMATION
*/
  Bundle request = makeRequestBundle("GET_PURCHASE_INFORMATION");
  request.putLong(REQUEST_NONCE, mNonce);
  request.putStringArray(NOTIFY_IDS, mNotifyIds);
  Bundle response = mService.sendBillingRequest(request);
  // Do something with this response.
}
```

GET_PURCHASE_INFORMATION 요청으로 인해 두 가지 broadcast intents 를 받을수 있다. 첫 번째로 Google Play Application 은 RESPONSE_CODE broadcast intent 를 보내는데, 이것은 요청에 대한 에러 정보와 상태 정보를 제공한다. 그리고 요청이 성공적이라면 PURCHASE_STATE_CHANGED broadcast intent 를 보낸다. 이 메시지는 상세 구매 정보가 JSON 문자열을 포함하고 있다.

9) 거래 정보 확인 알려주기

거래정보를 받았으면 이를 확인하였다고 Google Play Application 에 알려주기 위해서 CONFIRM_NOTIFICATIONS 메시지를 보내야 한다. 아래 코드는 이를 보인다.

```
/**
* Request type is CONFIRM_NOTIFICATIONS
*/
  Bundle request = makeRequestBundle("CONFIRM_NOTIFICATIONS");
  request.putStringArray(NOTIFY_IDS, mNotifyIds);
  Bundle response = mService.sendBillingRequest(request);
  // Do something with this response.
}
```

CONFIRM_NOTIFICATIONS 요청은 Bundle 객체를 반환하는데 이것은 요청의 상태정보를 나타내는 RESPONSE_CODE 키 와 요청에 대한 고유한 요청 식별자를 제공하는 REQUEST_ID 를 포함하고 있다. 그리고 이 요청을 받은 Google Play Application 은 요청에 대한 상태정보와 오류정보가 담긴 RESPONSE_CODE broadcast intent 를 보낸다.

10) 구매정보 복원하기

사용자의 구매 정보를 복원하기 위해서 RESTORE_TRANSACTIONS 메시지를 보내야 한다. 아래 코드는 이를 보인다.

```
/**
* Request type is RESTORE_TRANSACTIONS
*/
  Bundle request = makeRequestBundle("RESTORE_TRANSACTIONS");
  request.putLong(REQUEST_NONCE, mNonce);
  Bundle response = mService.sendBillingRequest(request);
  // Do something with this response.
}
```

RESTORE_TRANSACTIONS 요청은 Bundle 객체를 반환하는데 이것은 요청의 상태정보를 나타내는 RESPONSE_CODE 키 와 요청에 대한 고유한 요청 식별자를 제공하는 REQUEST_ID 를 포함하고 있다. GET_PURCHASE_INFORMATION 요청으로 인해 두 가지 broadcast intents 를 받을수 있다. 첫 번째로 Google Play Application 은 RESPONSE_CODE broadcast intent 를 보내는데, 이것은 요청에 대한 에러 정보와 상태 정보를 제공한다. 그리고 요청이 성공적이라면 PURCHASE_STATE_CHANGED broadcast intent 를 보낸다. 이 메시지는 상세 구매 정보가 JSON 문자열을 포함하고 있다.

11) BroadcastReceiver 생성

Google Play Application 은 billing 응답을 안드로이드 앱으로 보내기 위해서 broadcast intent 를 보낸다. 다음 그림은 이러한 broadcast intent 를 다루는 BroadcastReceiver 의 구현을 보인다.

```
public class BillingReceiver extends BroadcastReceiver {

  private static final String TAG = "BillingReceiver";

  // Intent actions that we receive in the BillingReceiver from Google Play.
  // These are defined by Google Play and cannot be changed.
  // The sample application defines these in the Consts.java file.
  public static final String ACTION_NOTIFY = "com.android.vending.billing.IN_APP_NOTIFY";
  public static final String ACTION_RESPONSE_CODE = "com.android.vending.billing.RESPONSE_CODE";
  public static final String ACTION_PURCHASE_STATE_CHANGED =
    "com.android.vending.billing.PURCHASE_STATE_CHANGED";

  // The intent extras that are passed in an intent from Google Play.
  // These are defined by Google Play and cannot be changed.
  // The sample application defines these in the Consts.java file.
  public static final String NOTIFICATION_ID = "notification_id";
  public static final String INAPP_SIGNED_DATA = "inapp_signed_data";
  public static final String INAPP_SIGNATURE = "inapp_signature";
  public static final String INAPP_REQUEST_ID = "request_id";
  public static final String INAPP_RESPONSE_CODE = "response_code";


  @Override
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (ACTION_PURCHASE_STATE_CHANGED.equals(action)) {
      String signedData = intent.getStringExtra(INAPP_SIGNED_DATA);
      String signature = intent.getStringExtra(INAPP_SIGNATURE);
      // Do something with the signedData and the signature.
    } else if (ACTION_NOTIFY.equals(action)) {
      String notifyId = intent.getStringExtra(NOTIFICATION_ID);
      // Do something with the notifyId.
    } else if (ACTION_RESPONSE_CODE.equals(action)) {
      long requestId = intent.getLongExtra(INAPP_REQUEST_ID, -1);
      int responseCodeIndex = intent.getIntExtra(INAPP_RESPONSE_CODE,
        ResponseCode.RESULT_ERROR.ordinal());
      // Do something with the requestId and the responseCodeIndex.
    } else {
      Log.w(TAG, "unexpected action: " + action);
    }
  }
  // Perform other processing here, such as forwarding intent messages to your local service.
}
```

12) nonces 생성

다음은 nonces를 확인하고 관리하고 생성하는 부분의 구현을 보인다.

```
private static final SecureRandom RANDOM = new SecureRandom();
private static HashSet<Long> sKnownNonces = new HashSet<Long>();

public static long generateNonce() {
  long nonce = RANDOM.nextLong();
  sKnownNonces.add(nonce);
  return nonce;
}

public static void removeNonce(long nonce) {
  sKnownNonces.remove(nonce);
}

public static boolean isNonceKnown(long nonce) {
  return sKnownNonces.contains(nonce);
}
```

2-6-2. 설계 개선 목표

In-app purchase 기능 구현에 대해서 사용성(적은 학습 곡선)과 생산성 향상이라는 두가지 목표를 가지고 개선하기로 했다.
개발자가 MarketBillingService 와의 메시지 흐름과 여러 컴포넌트들 그리고 보안에 관련된 로직을 잘 몰라도 쉽게 in-app purchase 를 구현할수 있도록 하고, 단순히 객체의 메소드를 호출하는 것 만으로 쉽게 사용할수 있도록 개선하고자 한다.
반복적인 json 파싱을 안하기 때문에 빠르고 편리하게 개발할수 있고, 초급 개발자가 활용하더라도 쉽게 개발할수 있도록 BillingService 와 Android Market Application 간의 메시지를 송수신하고, 보안에 관련된 로직을 AndroidBillingLibrary 에서 모두 수행하도록 로직을 분리하여 개발의 생산성과 편의성을 향상시켜 개선하는 것을 목표로 한다.

2-6-3. 아키텍쳐 개선을 위한 사전 활동

2-6-3-1. 아키텍쳐 개선의 시작

본 장에서는 개발의 생산성과 편의성을 향상시켜 개선할 수 있는 오픈소스를 적극적으로 활용하여 아키텍쳐를 개선하고자 한다.

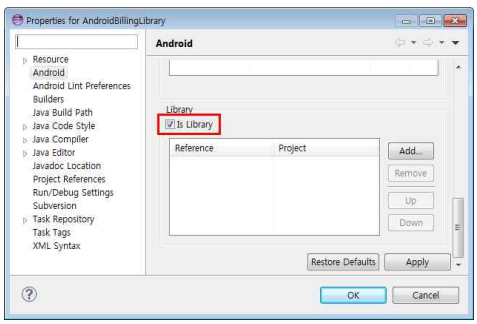
2-6-3-2. 활용할 오픈 소스 선택

본 문서에서는 이러한 안드로이드 In-app Purchase 기능을 구현하기 위해서 Android Billing Library 오픈소스 라이브러리를 선택하였다. Android Billing

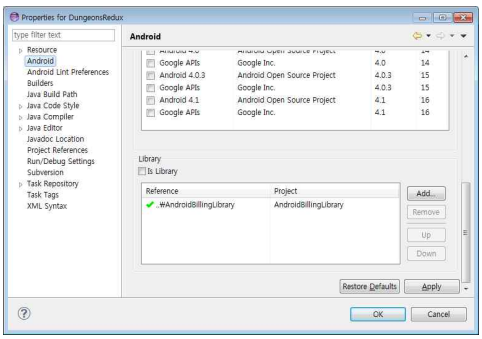
Library(https://github.com/robotmedia/AndroidBillingLibrary)는 복잡한 in-app billing 의 모든 기능을 보다 쉽게 구현할수 있도록 해주는 오픈소스 라이브러리이다.

가. Android Billing Library 설정

Android Billing Library 를 다운 받은 뒤 프로젝트의 properties 속성을 보게 되면, Library 용으로 만들어졌기 때문에 다음 그림과 같이 Is Library 가 체크되어있는 것을 확인할 수 있다. 그리고 in-app purchase 를 구현할 안드로이드 프로젝트를 생성하고, 프로젝트의 proterties 속성에서 Library 에서 참조할 라이브러리로 AndroidBillingLibrary 프로젝트를 추가해준다.



[그림 65. Android Billing Library 의 속성]



[그림 66. 샘플 앱의 속성]

그리고 AndroidManifest.xml 에 다음 permission 을 추가한다.

```
<uses-permission android:name="com.android.vending.BILLING" />
```

그리고 다음의 service 와 receiver 를 application 엘리먼트의 자식노드로 삽입한다.

```
<service android:name="net.robotmedia.billing.BillingService" />
<receiver android:name="net.robotmedia.billing.BillingReceiver">
    <intent-filter>
        <action  android:name="com.android.vending.billing.IN_APP_NOTIFY"
/>
        <action
android:name="com.android.vending.billing.RESPONSE_CODE" />
        <action
android:name="com.android.vending.billing.PURCHASE_STATE_CHANGED"
/>
    </intent-filter>
</receiver>
```

나. Android Billing Library 사용

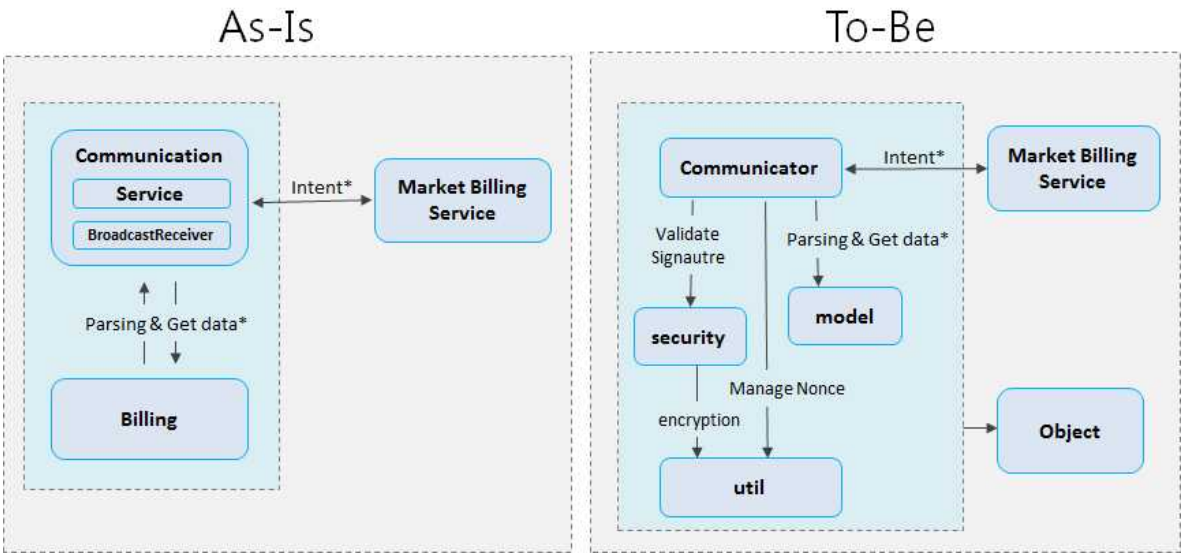
AbstractBillingActivity 는 in-app billing 과의 기본적인 통합을 위한 추상 액티비티이다. 만약 더 세밀한 컨트롤하기를 원한다면 BillingController 를 직접 사용하면 된다.
BillingController 는 Billing service 와의 더 높은 수준의 상호작용을 위한 기능들을 제공하고, 거래 관련 정보가 저장된 로컬 데이터베이스에 질의를 한다.
안드로이드 앱이 시작될 때 다음 코드와 같이 BillingController.IConfiguration 객체를 BillingController.setConfiguration 메소드를 호출하면서 매개변수로 설정해줘야 하고 메시지의 서명(signature) 확인 시 필요한 판매자 계정의 public key 와 거래정보들의 보안을 위해 필요한 salt 를 설정해야 한다.

```
public class Application extends android.app.Application {

@Override
public void onCreate() {
super.onCreate();
BillingController.setDebug(true);
BillingController.setConfiguration(
new BillingController.IConfiguration() {
public byte[] getObfuscationSalt() {
return new byte[] {41, -90, -116, -41, 66, -53, 122, -110, -127,
 -96, -88, 77, 127, 115, 1, 73, 57, 110, 48, -116};
}
```

```
public String getPublicKey() {
return "your public key";
}
});
}
}
```

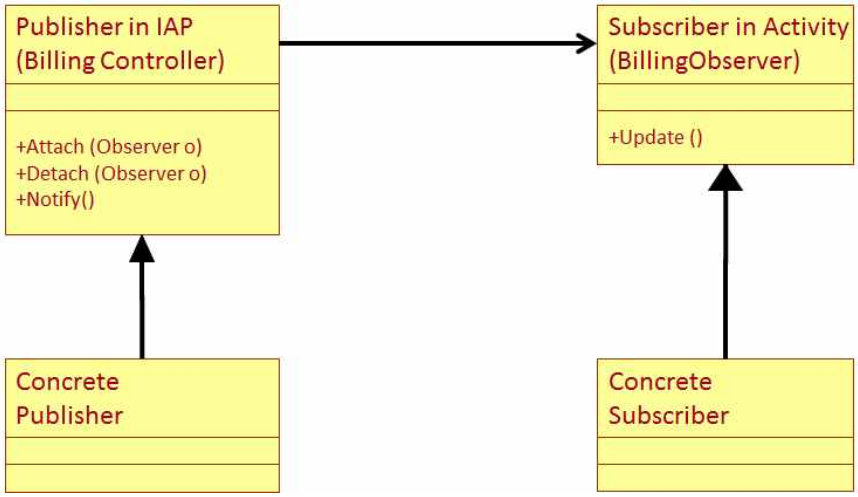
2-6-4. 아키텍쳐 개선 활동

보다 구조적이고 solid 한 아키텍쳐를 만들기 위해서, 기존 구조를 분석해서 문제점을 파악하여 간단한 구조부터 만들고자 했다.



[그림 67. 결재 컴포넌트의 현재 구조및 예상 구조]

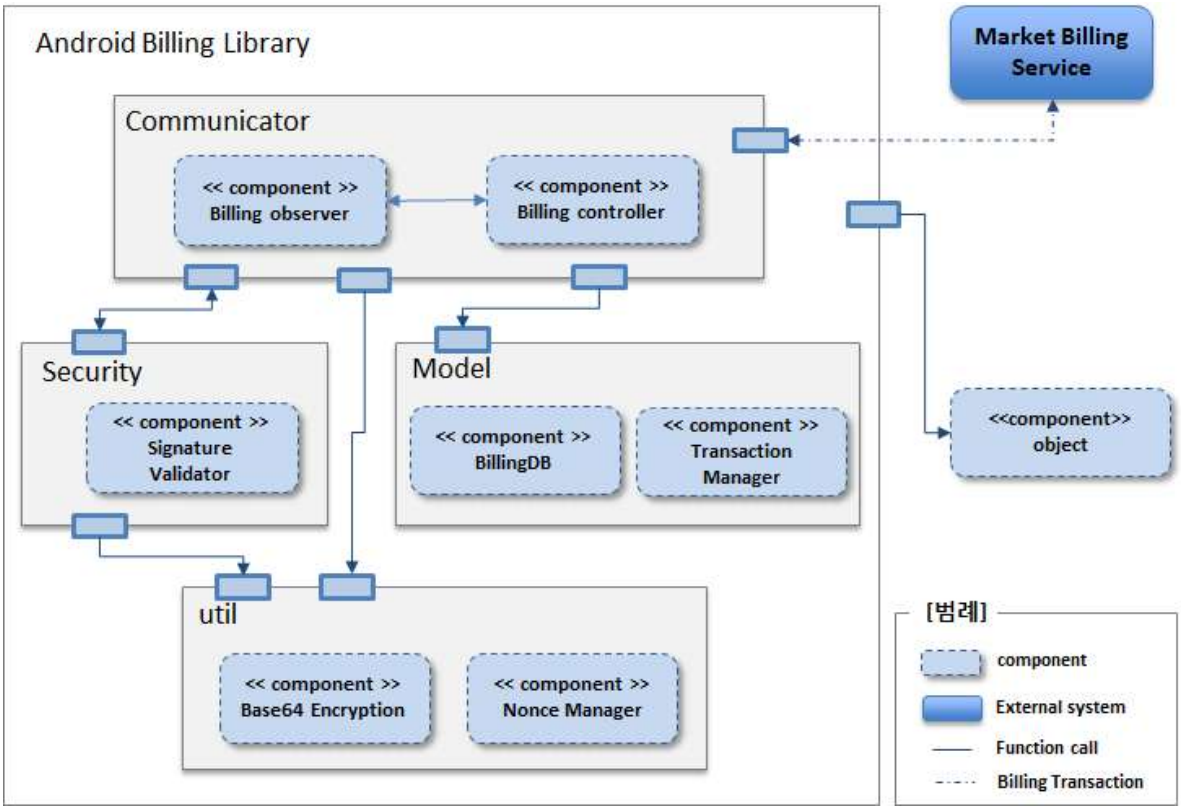
Android Billing Library 는 Observer 패턴을 이용하고 있다. Observer 패턴은 객체의 상태 변화를 관찰하는 관찰자들, 즉 옵저버들의 목록을 객체에 등록하여 상태 변화가 있을때마다 메서드등을 통해 객체가 직접 목록의 각 옵저버에게 통지하도록 하는 패턴이다. 주로 분산 이벤트 핸들링 시스템을 구현하는데 사용되며, Publish-subscribe pattern 으로도 알려져 있다.



[그림 68. Observer 패턴]

사용성을 강조하기 위해서 일반 사용자가 옵져버를 등록하고 간단한 함수를 호출하여 사용할수 있도록 되어있다. 기존 레거시 시스템의 자유도는 보장하면서, 시스템 사용자에게는 높은 개발 생산성을 보장하기 위해서 Observer 패턴을 적용되어있다.

2-6-5. 아키텍쳐 개선 결과



[그림 69. 개선된 결재 라이브러리의 구조]

기존 Market Billing Service 와의 연동을 위해, 정해진 메시지를 송수신하고, json 데이터를 생성하거나, 파신하는 코드들이 삽입되어 가독성과 생산성이 저해되었다.

이러한 문제를 해결하기 위해, Android Billing Library 는 다음과 같이 컴포넌트들을 구성하였다.

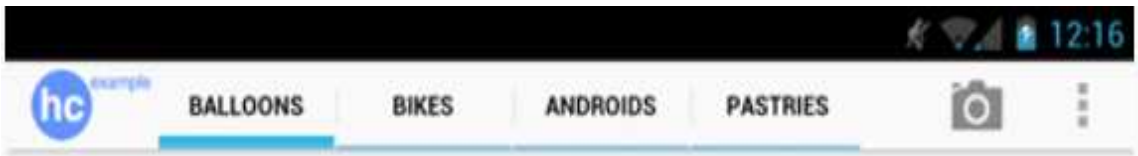
- Communicator: MarketBillingService 와의 메시지 송수신과 json 데이터 통신을 전담한다. 개발자는 메시지 송수신 과정과 json 데이터를 알 필요가 없다.
- Security: MarketBillingService 와의 메시지 송수신 과정에서 Signature 를 검증하는 역할을 한다.
- Model: 개발자가 쉽게 객체로 핸들링 할수 있게 하기 위해서 거래정보와 관련된 json 데이터를 분석한뒤 객체로 변환하고, 거래정보를 DB 에 저장하거나 DB 에서 가져오는 역할을 한다.
- Util: in-app purchase 거래정보의 무결성을 보장하기 위한 Nonce 를 생성하거나, Base64 인코딩과 디코딩을 담당한다.

Android Billing Library 에 대한 자세한 예제는 별첨 6 에서 확인할 수 있다.

2-7. ActionBar

2-7-1. 개요 및 배경

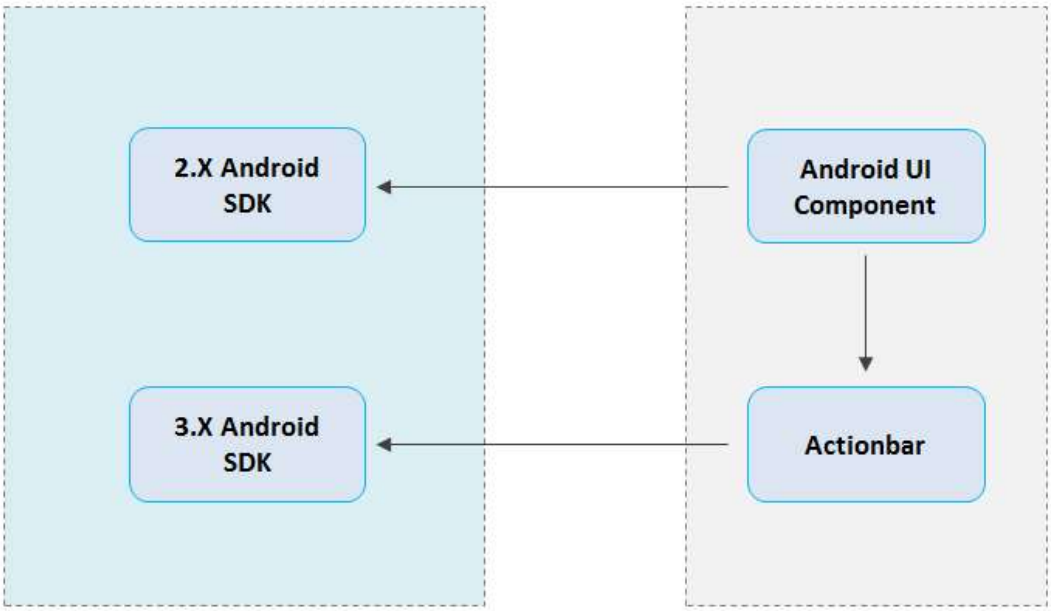
액션바(Actionbar)는 안드로이드 버전 3.0(API level 11)부터 새롭게 추가된 UI 컴포넌트로 기존 안드로이드 애플리케이션의 타이틀바(Title bar)를 대체한다. 기존 타이틀바가 단순히 제목만을 표시하거나 간단한 정보만을 표시할 수 있던 것에 반해 액션바는 메뉴제공, 액션 아이템을 통한 단축메뉴제공 등 다양한 기능을 포함하고 있다.



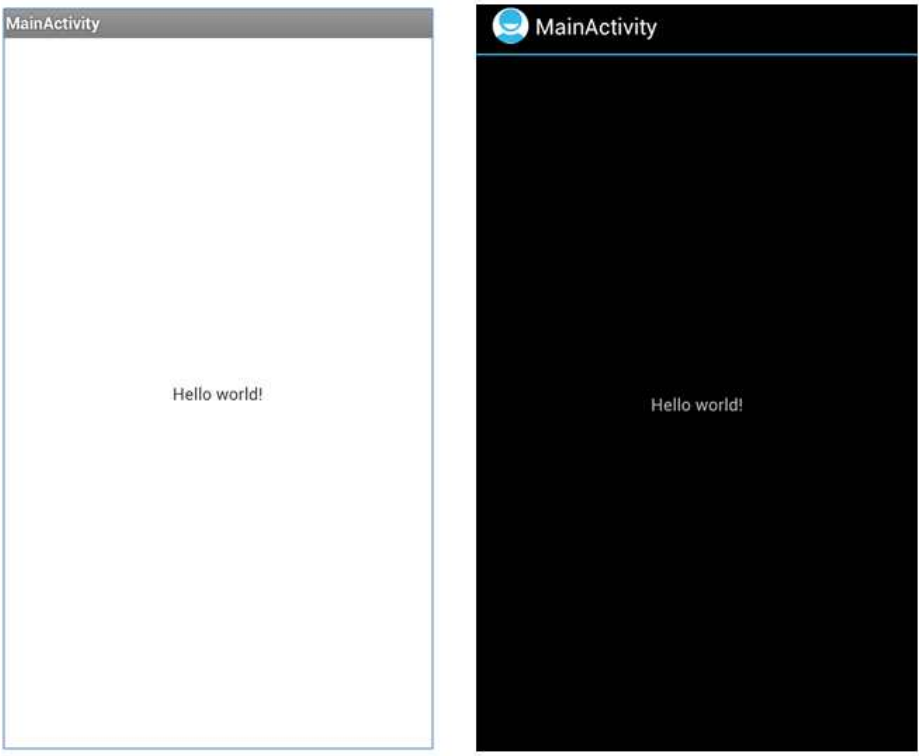
[그림 70. Honeycomb Gallery 앱의 액션바 형태(Landscape)]

구글에서는 액션바를 디자인의 표준으로 삼고 개발자들에게 이를 권장하고 있으며, 액션바를 이용해 기존의 테마에 보다 직관적이고 간결한 UI 를 가지고 있는 Theme.Holo

계열 테마를 제공한다. 하지만 액션바는 안드로이드 버전 3.0(API level 11)부터 새롭게 추가된 UI 컴포넌트이기 때문에 하위 버전에선 액션바를 사용하지 못하는 문제점이 있다.



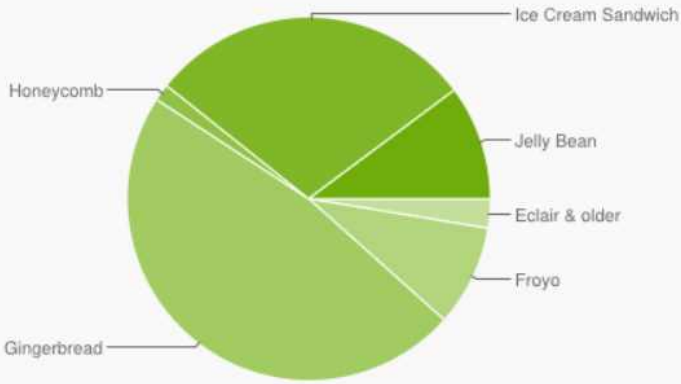
[그림 71. 안드로이드 버전 3.0 이하와 3.0 이상 에서의 구조]



[그림 72 <왼쪽>안드로이드 버전 2.3 기본 액티비티, <오른쪽>안드로이드 버전 4.0 기본 액티비티₩

위의 구조처럼 안드로이드 버전 3.0 이하에서는 액션바를 사용하지 못하고 일반적인 UI 컴포넌트만을 이용해야하며, 3.0 이후의 버전에서는 액션바를 사용할 수 있다. 따라서 만약 안드로이드 버전 3.0 이하의 기기에도 서비스가능한 애플리케이션을 만들기 위해서는 액션바를 사용한 레이아웃과 사용하지 않은 레이아웃 두가지 경우를 모두 작성해서 배포해야한다는 문제점이있다.

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.4%
2.2	Froyo	8	9.0%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	47.4%
3.1	Honeycomb	12	0.4%
3.2		13	1.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.1%
4.1	Jelly Bean	16	9.0%
4.2		17	1.2%



[그림 73. 안드로이드 버전 분포현황 (2013.1 월)]

2-7-2. 설계 개선 목표

위에서 언급한것 처럼 구글에서 디자인 표준으로 삼고있는 액션바는 안드로이드 버전 3.0 부터 새롭게 추가된 UI 컴포넌트이다. 따라서 하위 버전에서는 사용할 수 없으며 이 때문에 액션바를 사용하지 않는 레이아웃을 추가적으로 구현해야만 했다.

본 문서에서는 이 같은 문제점을 해결하고자 한다. 안드로이드 버전 3.0 이하에서도 액션바를 사용할 수 있게 함으로써, 하나의 레이아웃으로 모든 안드로이드버전에서 실행하는 것을 목적으로 한다.

2-7-3. 아키텍쳐 개선을 위한 사전 활동

2-7-3-1 아키텍쳐 개선의 시작

본 장에서는 하위버전에서의 액션바 사용을 위해 오픈 소스를 활용하여 기존 문제점을 해결하고자 한다.

2-7-3-2 활용할 오픈 소스 선택 및 적용

하위 버전에서의 액션바 사용을 위해서 액션바셜록(Actionbar Sherlock) 오픈소스를 활용하였다. 액션바셜록(Actionbar Sherlock)은 API level 11 이하에서도 액션바를 사용하기 위한 하위호환 라이브러리로 간단하게 하위의 버전에서도 액션바를 사용할 수 있게 해준다. 액션바셜록은 jar 로 묶여있는 라이브러라기 아니라 소스코드형태로 배포되는 형태기 때문에 커스텀이 가능하다는 특징도 가지고 있으며 소스코드를 받아서 사용해야한다. 이는 액션바셜록 공식홈페이지에서 받을 수 있다 (http://actionbarsherlock.com).

[그림 74 액션바셜록 공식홈페이지 (http://actionbarsherlock.com)]

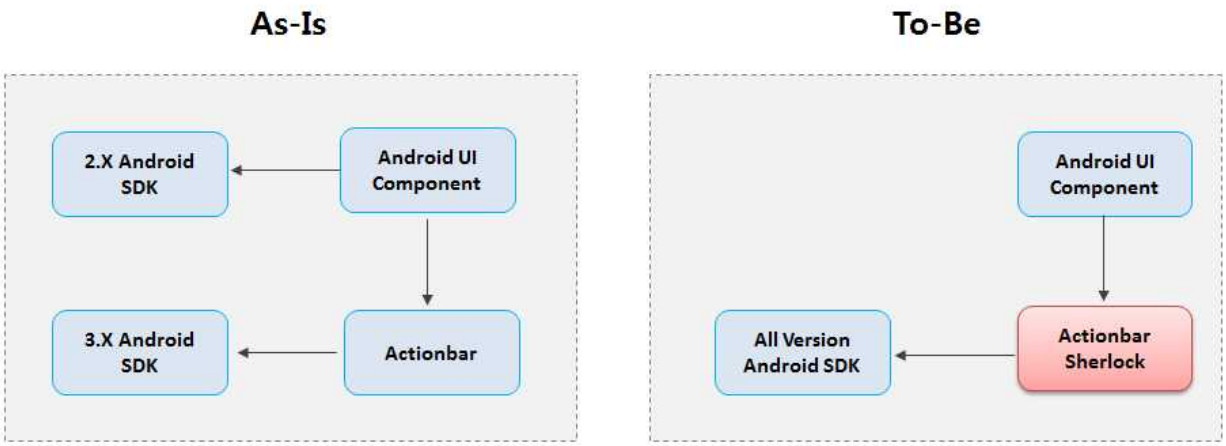
액션바셜록을 이용하기 위한 액션바셜록 라이브러리 프로젝트 생성하기 방법은 별첨 7 에서 다룬다.

액션바셜록을 이용하면 안드로이드 버전 3.0 이하에서도 액션바를 이용할 수 있으며 동일한 UI 를 얻을 수 있다.

2-7-4. 아키텍쳐 개선 활동

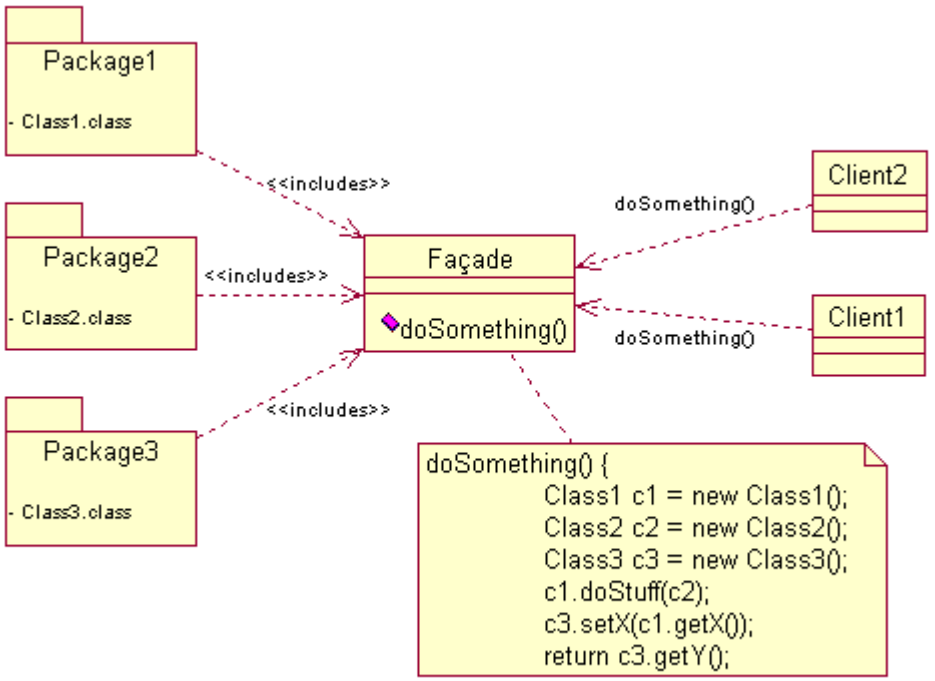
2-7-4-1. 구조 개선

하위 버전에서 액션바를 사용하지 못하는 문제점을 해결하기 위해서 액션바셜록을 사용하였다. 아래 그림에서 볼 수 있듯이 기존(As-Is)에는 안드로이드 버전 3.0 이상에서만 액션바를 사용할 수 있었으며 하위버전에서는 사용하지 못했다. 액션바셜록을 이용한 결과(To-Be) 안드로이드 버전에 상관 없이 항상 액션바를 사용하여 동일한 UI 로 만들 수 있게 되었다.



[그림 75. 액션바의 현재 구조및 개션될 예상 구조]

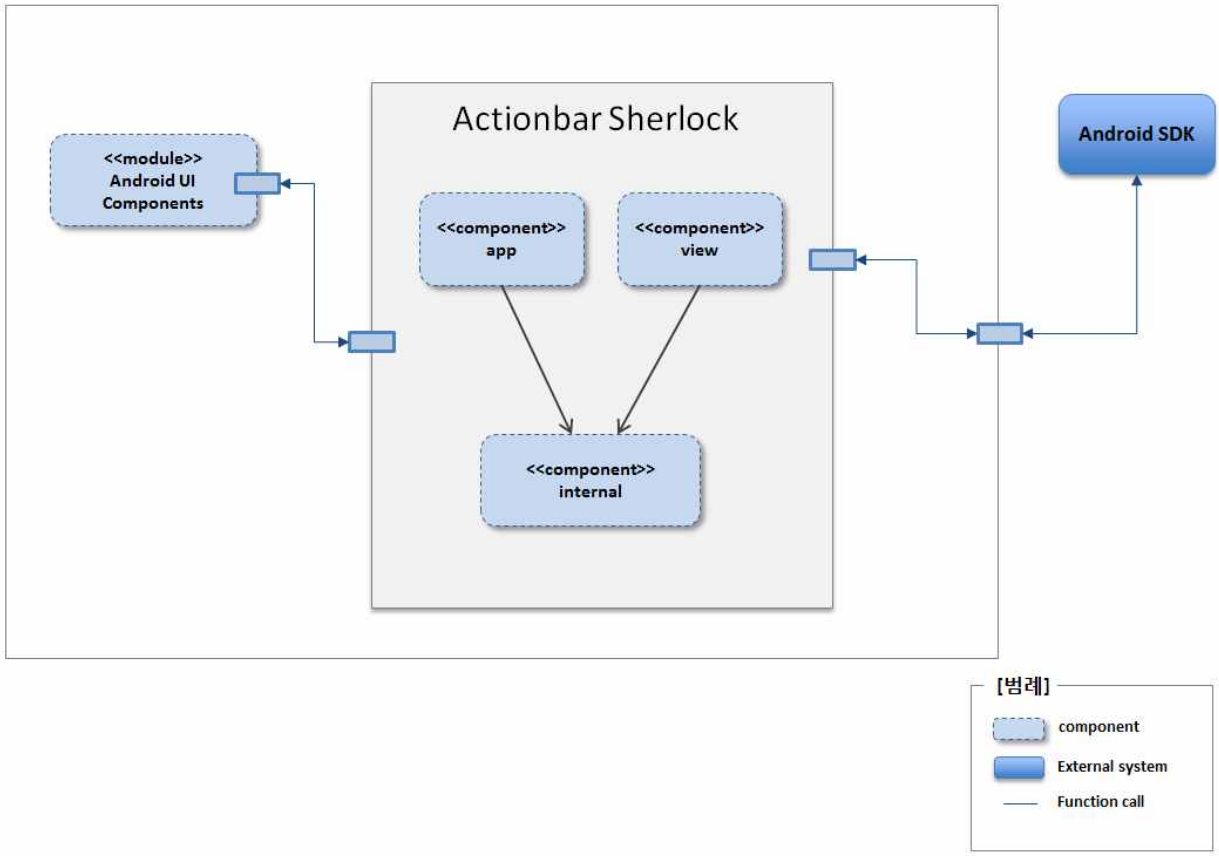
액션바셜록(Actionbar Sherlock)에서는 래퍼 퍼사드 패턴을 이용하여 모든 버전에서 쉽게 액션바를 사용할 수 있도록 하였다.



[그림 76. Wrapper Façade 패턴]

결국 사용자는 액션바셜록에서 제공하는 메소드 들만 고려하여 코드를 작성하면 모든 버전에서 동일한 UI 를 가지게 되는것이다.

2-7-5. 아키텍쳐 개선 결과



[그림 77. 개선된 액션바의 구조]

기존엔 하위버전에서 액션바를 사용하지 못했기 때문에 해당 버전에 맞는 UI 를 전개시켜줘야 하는 문제점이 있어서 생산성이 저해되었다.

이러한 문제를 해결하기 위해, 오픈소스 액션바셜록을 이용하여 다음과 같이 컴포넌트들을 구성하였다.

- Android UI components: menu, layout, fragment 등 일반적으로 안드로이드에서 사용하는 UI 컴포넌트들의 모음으로 UI 를 구성할 때 필요한 컴포넌트들이 존재한다.
- Actionbar Sherlock: 하위버전 Android UI component 에 존재하지 않는 액션바 컴포넌트들을 가지고 있다. View 컴포넌트에는 메뉴, 액션프로바이더(action provider) 등이 존재하며, app 컴포넌트에는 기존의 Fragment, Activity 들을 액션바 형태에 맞춘 SherlockFragmentActivity, SherlockListActivity 등이 존재한다.

결국 기본 Android UI component 와 Actionbar Sherlock 의 컴포넌트를 이용하여 모든 버전에서 액션바를 사용할 수 있는 구조이다.

2-7-5-1. 개발 생산성 개선

액션바셜록을 사용한 경우와 그렇지 않은 경우를 비교했을 때 생산성 개선이 이루어졌다. 기존의 경우 하위버전일 경우 액션바가 아닌 레이아웃으로 레이아웃을 작성해야 했으며, 안드로이드 버전(API Level)에 따라 해당 버전에 맞는 레이아웃을 전개시켜주어야 했다. 액션바셜록을 사용한 경우는 Activity 대신 SherlockActivity 만 extends 해주면 버전과 상관없이 액션바 레이아웃을 사용할 수 있다.

- 기존의경우(액션바셜록 사용안함)

```
public class MainActivity extends Activity {
  public static final int mCurrentApiVersion =
android.os.Build.VERSION.SDK_INT;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if(mCurrentApiVersion >= 11) { /* use actionbar*/
          setContentView(R.layout.activity_main);
        } else { /* don't use actionbar*/
          setContentView(R.layout.activity_main_no_actionbar);
        }
    }

}
```

그림 17 액션바셜록 사용안한 경우

- 액션바셜록 사용

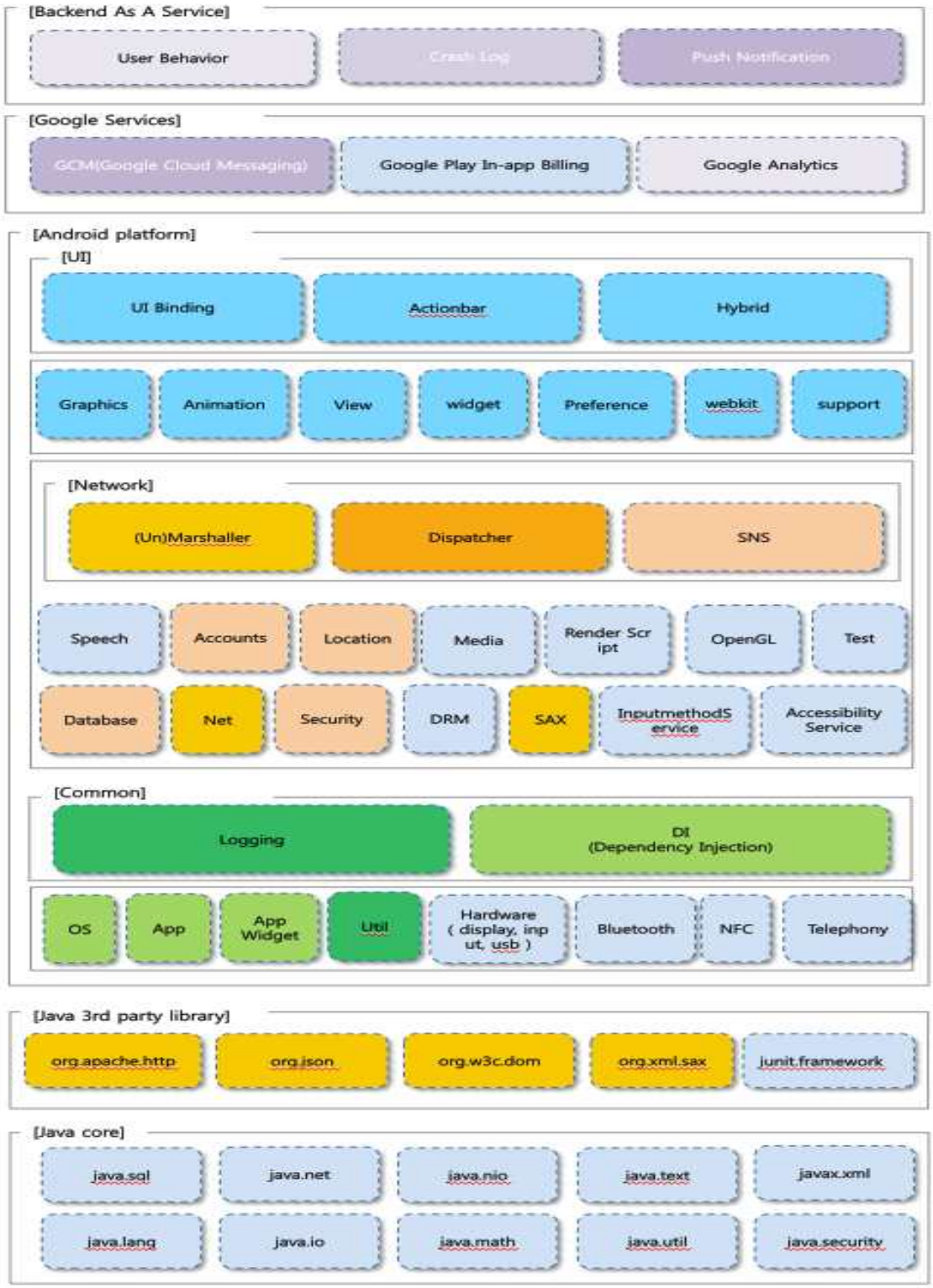
```
public class MainActivity extends SherlockActivity {
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main_activity);
    }
}
```

그림 18 액션바셜록 사용한 경우

액션바 컴포넌트들에 대해서는 별첨 8 을 통해서 좀 더 자세히 소개하였다.

2-8. 안드로이드 오픈소스 어플리케이션 블록 전체 아키텍처



[그림 78. 안드로이드 오픈소스 어플리케이션 블록]

본 문서는 크게 4 계층에 대한 아키텍쳐 블록을 선정하고, 어떻게 설계가 되었는지 자세히 설명하였고, 오픈소스를 사용하여 실제 프로젝트에 활용할 수 있는 부분은 별첨을 통해 다루었다.

- 유저 인터페이스 (User Interface)
  UI 개발시 안드로이드의 제약사항을 극복하기 위해 아래와 같은 오픈소스 어플리케이션 블록을 본 문서에서 설명하였다.

  - 단편화 문제를 해결하기 위한 Wrapper Façade 패턴을 이용한 Actionbar
  - 사용자마다 개인화된 뷰를 가질수 있는 MVC 패턴을 이용한 UI Binding 기법
  - 제한된 화면에서 계층화된 화면을 보여 줄수 있는 Side Navigation 패턴

- 네트워크 (Network)
  모바일 특성상 제한된 리소스로 인해, 외부 서비스와 연동이 필수이지만 안드로이드는 네트워크 라이브러리의 기능이 빈약하다. 이 문서는 다음과 같은 네트워크 어플리케이션 블록의 아키텍쳐와 실 사용법을 설명하고 있다.
  - 복잡한 프로토콜 파싱없이 쉽게 REST (XML/JSON)을 객체로 변환하는 Marshaller/Unmarshalleer (Composite Message 패턴)
  - 앱 홍보와 가입없이 사용자를 쉽게 확보하기 위해 필요한, SNS 제어 라이브러이인 fHalo (Wrapper Façade 패턴과 코드 사용성 시나리오 기법 소개)

- 공용부 (Common)
  안도로이드는 컴포넌트간의 메세지와 간단한 속성만 설정을 할수 있으며, 로그 정보를 Eclipse 와 같은 통합 개발 환경(IDE)에 연결해야만 볼수 있는등 실제 서비스 배포시 필요한 여러 공용부 기능이 빠져있다. 본 문서는 다음과 같은 공용부 기능들을 설명하고 있다.

  - 안드로이드 내장 로그인 logcat 에서 제공하지 않는 일반적인 로그및 크래쉬 정보를 SD 카드에 저장하는 로그 핸들러 구성 기법
  - 컴포넌트를 런타임시에 할당및 대체할 수 있는 컴포넌트 설정부 (Component Configurator) 를 구성하는 기법과 구성 원리

- 서비스로써의 백엔드 (BaaS)

구글이 제공하는 부족한 백엔드 기능 (푸쉬, 사용자 통계, 오류 리포트)을 극복하기 위한 기법들을 소개한다.

- 로그및 크래쉬(Crash) 정보를 수집하여 그래픽으로 비주얼라이제이션 하는 Log 수집기와 웹 리포터 기능 구축 방법및 구성 원리
- 배포한 앱의 실 사용 수치/국가별 배포량/ 행동 패턴등을 분석하는 기법 설명
- 인 앱 결재(In-App-Purchase)를 간단히 할수 있는 빌링 (결재) 라이브러리를 구축하는 기법과 구성 원리

[별첨]

[별첨1] Simple Framework 참고 예제

1. XML 형식 예제

```
<?xml version="1.0" encoding="UTF-8"?>
  <wid>
      <header>
          <tm>201212021800</tm>
      </header>
      <body>
          <location province="11B00000" city="11B10101">
          <province code="11B00000">서울 · 인천 · 경기도</province>
          <city code="11B10101">서울</city>
              <data>
                  <numEf>2</numEf>
                  <tmEf>2012-12-04</tmEf>
                  <wf>구름조금</wf>
                  <tmn>-6</tmn>
                  <tmx>0</tmx>
                  <reliability>보통</reliability>
              </data>
              <data>
                  <numEf>3</numEf>
                  <tmEf>2012-12-05</tmEf>
                  <wf>구름많고 눈</wf>
                  <tmn>-6</tmn>
                  <tmx>1</tmx>
                  <reliability>보통</reliability>
              </data>
          </location>
      </body>
  </wid>
```

위의 예제는 기상청 사이트에서 제공하는 XML 형식의 날씨 데이터이다. XML 형식의 데이터는 <wid>, <header>, <title>, <tm> 등의 start tag 로 시작되고, </wid>, </header>,

</title>, </tm> 등의 end tag 과 쌍으로 이뤄져 있음을 확인할 수 있다. 또한 start tag 와 end tag 안에는 Text 형식의 데이터 값이 들어가 있음을 알 수 있다.

2. XML 형식 데이터 파싱 예제 코드

```
<?xml version="1.0" encoding="UTF-8"?>
<categoryList>
  <categories>
        <category>
              <id>1</id>
              <name>AA</name>
              <date>2012-11-27</date>
        </category>
        <category>
              <id>2</id>
              <name>BB</name>
              <date>2012-11-28</date>
        </category>
  </categories>
</categoryList>
```

위의 같은 XML 이 주어졌을 때 개발자들은 SAX(Simple APT for XML) 이나 DOM (Document Object Model) Parser 등 다양한 방식을 이용하여 XML 을 오브젝트 형식으로 변환한다.

```
CategoryList categoryList = new CategoryList();
Category category = null;

XmlPullParser parser = getResources().getXml(R.xml.category);

int eventType = parser.getEventType();
String tag = "" ;
while (eventType != XmlPullParser.END_DOCUMENT) {
        switch (eventType) {
        case XmlPullParser.START_TAG:
                tag = parser.getName();
                if(tag.equals("category")){
                        category = new Category();
                }
                break;
        case XmlPullParser.TEXT:

                if(tag.equals("id")){
                        category.setId(Integer.parseInt(parser.getText()));
                }else if(tag.equals("name")){
                        category.setName(parser.getText());
                }else if(tag.equals("date")){
                        category.setDate(parser.getText());
                }
                break;
        case XmlPullParser.END_TAG:
                String endTag = parser.getName();
                if(endTag.equals("category")){
                        categoryList.categories.add(category);
                }
                break;

        default:
                break;
```

```
            }
            eventType = parser.next();
        }
```

위의 예제는 XML 형식의 데이터를 안드로이드 플랫폼에서 제공하는 XML Pull Parser 방식을 통해 데이터를 파싱한 예제 코드이다.

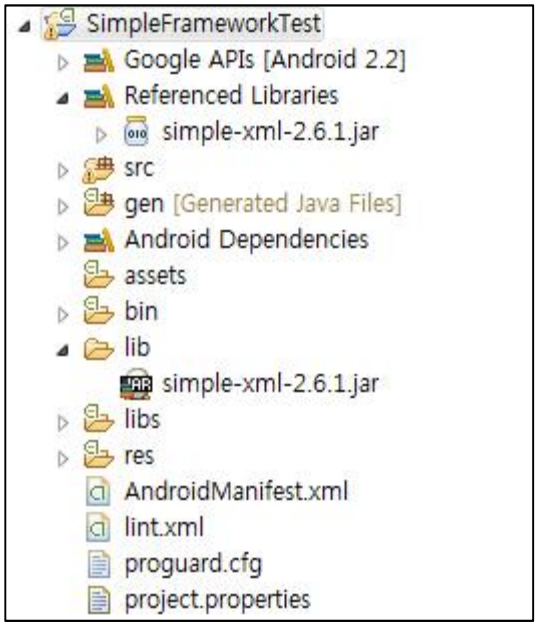
XML Pull Parser 방식은 위와 예제 코드와 같이 한줄 한줄 코드를 읽어가며 category, id, name, date 등과 같은 태그를 읽어줘야 한다.

- START_TAG : XML 데이터 중 시작 태그를 인식하였을 때 (ex. <tag>)
- TEXT : XML 데이터 중 텍스트를 인식하였을 때 (ex. <tag> **text** </tag>)
- END_TAG : XML 데이터 중 종료 태그를 인식하였을 때 (ex. </tag>)
- END_DOCUMENT :XML 파일의 끝에 도달하였을 때

3. 심플 프레임워크를 통한 XML 데이터 파싱 예제 코드

1) 해당 Library 저장

Simple Framework 의 공식 홈페이지에서 제공하는 라이브러리를 다운로드 받은 후, 해당 jar 파일을 Android 프로젝트에 빌드시킨다.
(http://simple.sourceforge.net/download.php)



[그림 1. Simple Framework 설정]

2) XML 구조에 맞는 어노테이션을 이용하여 객체 클래스를 생성

위의 XML 데이터를 보면 categoryList, categories, category, id, name, date 의 태그를 확인할 수 있다. 이 XML 구조를 분석하면 categoryList 태그는 categories 태그를 포함하고 있으며, categories 태그 안에는 category 라는 태그의 배열로 이루어져 있음을 확인할 수 있다. 이 구조를 클래스로 작성하면 categoryList 클래스와 같다.

또한 category 태그는 id, name, date 의 태그와 value 값으로 이루어져 있으며 이 구조를 코드로 작성하면 아래의 Category 클래스와 같다.

일반적으로 태그와 클래스 명을 일치시켜줘야 하지만, name 속성값을 통해 태그 값을 입력해 줄 수도 있다. **(Ex. @Element(name = "id"))**

이와 같이 XML 구조를 분석한 다음, 아래와 같이 해당 라이브러리를 import 시킨 후 @Root, @Element, @ElementList 등 각각에 맞은 어노테이션을 통해 클래스 코드를 작성한다.

```
import org.simpleframework.xml.ElementList;

import org.simpleframework.xml.Root;

import org.simpleframework.xml.Element;
```

```
@Root
```

```
public class CategoryList {


  @ElementList
  public List<Category> categories = new ArrayList<Category>();
}
```

```
@Root
public class Category {


  @Element(required=false)
  private int id;


  @Element(required=false)
  private String name;


  @Element(required=false)
  private String date;


  public int getId() {
            return id;
  }


  public void setId(int id) {
            this.id = id;
  }
```

```
      ......

    }
```

3) 객체로 변환

심플 프레임워크 라이브러리에서 제공하는 메소드를 통해 xml 형식의 데이터를 객체로 변환한다.

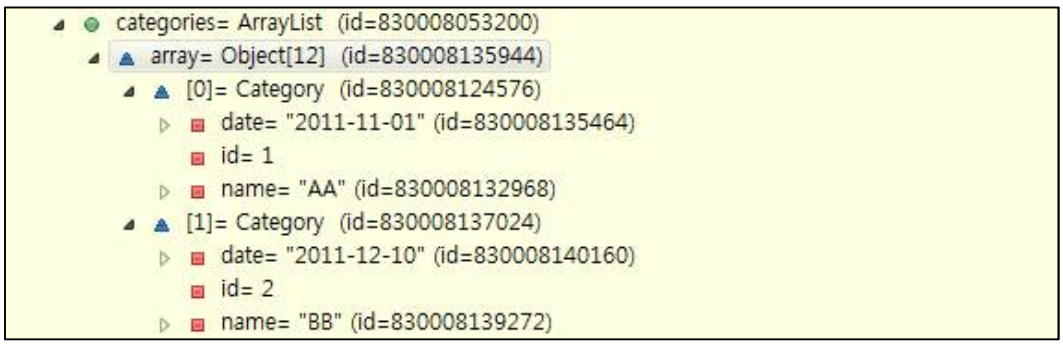
```
Serializer serializer = new Persister();

Reader reader = new StringReader(xml);

CategoryList categoryList = serializer.read(CategoryList.class, reader, false);
```

해당 함수를 사용하기 위해서는 아래와 같은 라이브러리를 import 시켜줘야 한다.

```
import org.simpleframework.xml.Serializer;

import org.simpleframework.xml.core.Persister;
```

4) 결과값 확인

Logcat 또는 화면를 통해 데이터가 올바르게 읽혀졌는지 확인한다.



4. json 형식 예제

```
{
  "id": "100000000000000",
  "name": "MiKyung Kang",
```

```
        "first_name": "MiKyung",
        "albums": {
         "data": [
          {
           "id": "100000000000000",
           "updated_time": "2012-10-12T11:53:40+0000",
           "created_time": "2011-04-23T12:09:31+0000"
          },
          {
           "id": "100000000000000",
           "updated_time": "2012-07-15T04:06:02+0000",
           "created_time": "2012-04-02T08:30:21+0000"
          }
         ]
        }
       }
```

위의 예제는 Facebook 개발자 사이트에서 제공하는 Json 형식의 사용자 데이터이다. “id” ,“name” “first_name” 의 name 값을 확인할 수 있으며, “100000000000000”:, “MiKyung Kang”, “MiKyung” 등의 Value 값도 확인해 볼 수 있을 것이다. 또한 “albums”라는 네임 값 안에는 대괄호를 통해 배열형식의 여러 데이터가 들어가 있음을 확인할 수 있다. (http://developers.facebook.com/tools/explorer 참고)

5. JSON 형식 데이터 파싱 예제 코드

```
CategoryList categoryList = new CategoryList();
Category category = null;

JSONObject jsonObject = new JSONObject(json);
JSONArray array = jsonObject.getJSONArray("categories");

for(int index =0 ; index < array.length() ; index++ ){
        category = new Category();

        JSONObject categoryObject = array.getJSONObject(index);
```

```
            category.setId(categoryObject.getInt("id"));
            category.setName(categoryObject.getString("name"));
            category.setDate(categoryObject.getString("date"));

            categoryList.categories.add(category);
        }
```

위의 예제는 JSON 형식의 데이터를 Android SDK 에서 제공하는 JSON Parser 방식을 통해 데이터를 파싱한 예제 코드이다.
JSON Parser 방식 또한 JSON 형식의 데이터를 읽어들 인 후, 구문을 분석하여 JSONArray 및 JSONOjbect 에 데이터를 삽입한다. 또한 id, name, date 등과 같은 key 값을 분석하여 각각의 Name 값에 해당하는 Value 값을 파싱하는 방식이다.

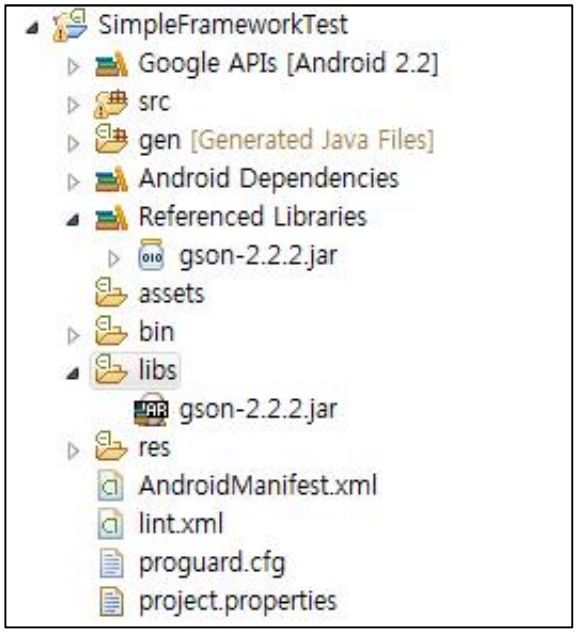
Json Parser 는 XMl Pull Parser 방식과 달리, JSONArray 클래스 및 JSONObejct 클래스를 통해 JSON 데이터의 배열 및 객체 정보를 저장할 수 있다.

- JSONArray : JSON 데이터의 배열을 읽어 들인다. 생성자로 JSON 문자열을 전달하면 파싱 후 배열 형식으로 데이터를 저장하고 있다.
- JSONObject : JSON 데이터의 객체 정보를 읽어 들인다.

6. GSON 프레임워크를 통한 JSON 데이터 파싱 예제 코드

1) 해당 Library 저장

Gson Framework 의 공식 홈페이지에서 제공하는 자바 라이브러리를 다운로드 받은 후, 해당 jar 파일을 Android 프로젝트에 빌드시킨다.
(http://code.google.com/p/google-gson/)



[그림 2. GSON 프레임워크 설정]

2) Json 구조에 맞는 객체 클래스를 생성

위의 Json 데이터를 보면, “categories”, “id”, “name”, “date” 등의 Name 값을 확인할 수 있다. 이 Json 구조를 분석하면 “categories” Name 안에는 “id,”, “name”, “date”의 정보가 들어있는 배열로 이루어져 있음을 알 수 있다. 때문에 CategoryList.class 는 Category.class 구조를 지닌 배열로 코드를 작성한다. 또한 Category.class 는 실제 데이터를 저장 할 “id,”, “name”, “date” 등의 필드로 이루어진 구조로 코드를 작성한다.

```
public class CategoryList {


  public List<Category> categories = new ArrayList<Category>();

}
```

```
public class Category {

  private int id;

  private String name;

  private String date;

  public int getId() {
        return id;
  }

  public void setId(int id) {
        this.id = id;
  }

  ......

}
```

3) 객체로 변환

Gson 프레임워크 라이브러리에서 제공하는 함수를 통해 json 형식의 데이터를 객체로 변환한다.

```
import com.google.gson.Gson;
```

```
Gson gson = new Gson();

CategoryList categoryList = gson.fromJson(json, CategoryList.class);
```

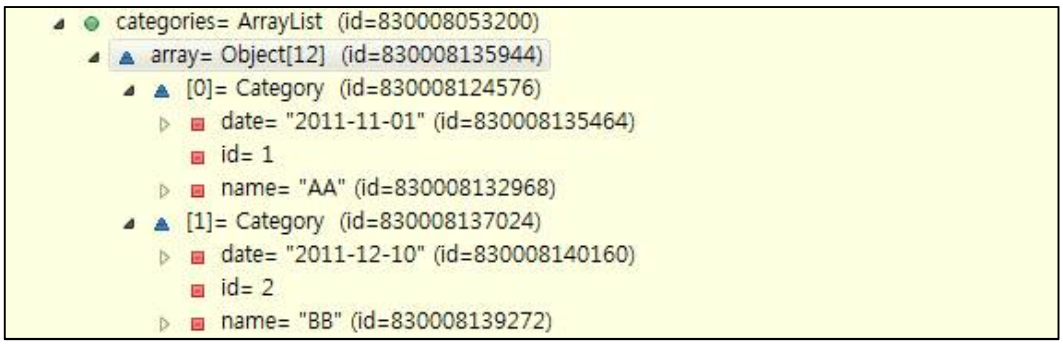
위의 코드는 json 데이터를 객체로 변환해주는 예제 코드이다. Gson 객체를 생성해 준 후, Json 데이터를 파싱할 때는 fromJson 함수를 사용하면 되기 때문에, Gson 프레임워크는 심플 프레임워크에 비해 직관적인 함수를 가지고 있으며, 사용법도 매우 간단하다.

- fromJson : Json 데이터를 객체로 변환해주는 함수
- toJson : 객체를 Json 형식의 문자열로 변환해주는 함수

Json 데이터를 파싱하는 함수는 toJson 을 통해 객체화 할 수 있으며, 그 외에도 fromJson 함수를 통해서 객체를 JSON 형식의 스트링으로 변환할 수 있다.

4) 결과 값 확인

Logcat 또는 화면을 통해 데이터가 올바르게 읽혀졌는지 확인한다.



[별첨2] microlog4android 설치 및 설정 방법

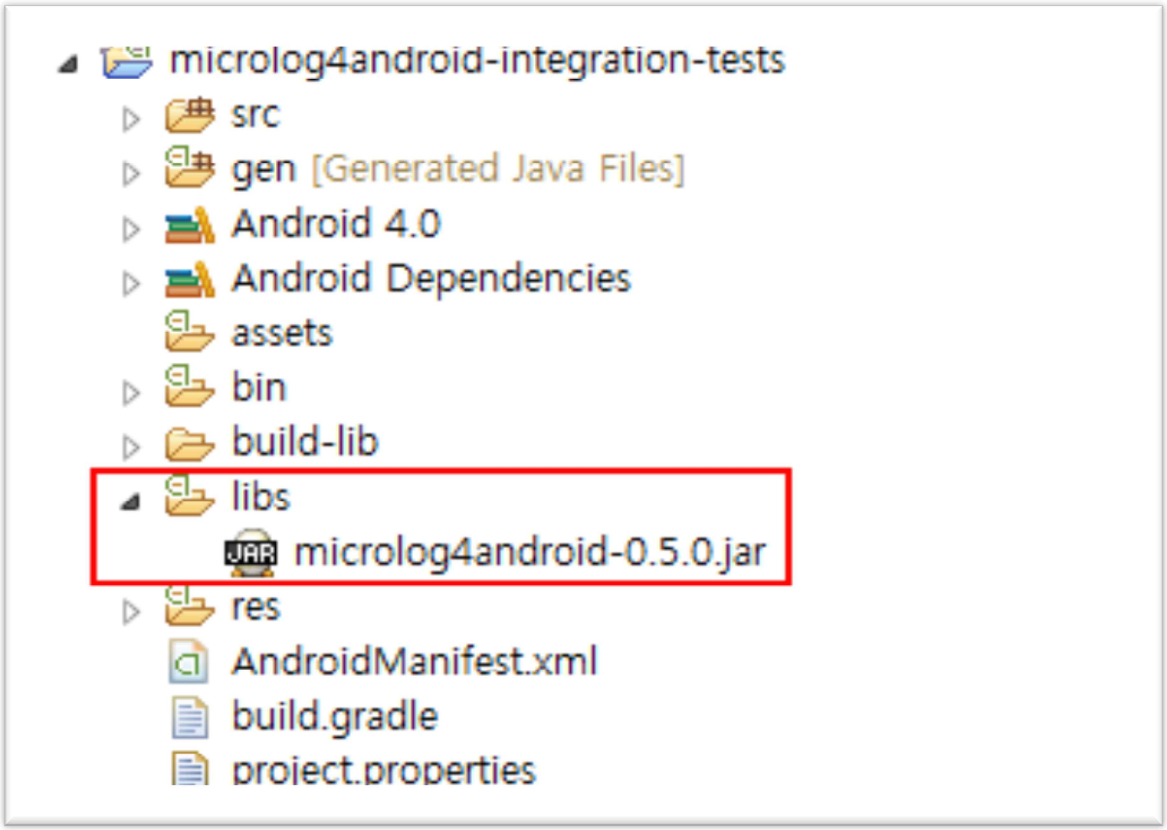
A. Hello World

microlog4android 를 이용하기 위해서는 다음의 준비사항이 필요하다.

- 안드로이드 개발환경이 갖추어진 이클립스(Eclipse) IDE
- 최신 microlog4android 라이브러리

B. microlog4android 설정

이클립스 IDE 에서 기본 안드로이드 프로젝트를 생성하고, 프로젝트의 "libs" 폴더에 microlog4android library 를 설치 한다.



[그림 3. microlog4android 설정]

그리고 AndroidManifest.xml 에 android.permission.WRITE_EXTERNAL _STORAGE 를 추가한다.

```
<?xml version="1.0" encoding="utf-8"
<manifest xmlns:android="http://schemas.android.com/apk/res/android"    ... >
...
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<application
    android:name="com.malangstudio.alarmmonhd.Application"
    android:icon="@drawable/icon" android:label="@string/app_name" >
  ...
</application>
...
</manifest>
```

C. microlog4android 관리 클래스 생성

microlog4android 라이브러리를 추가하고, AndroidManifest.xml 에 android.permission.WRITE_EXTERNAL _STORAE 를 추가하였다면, 로그를 기록하는 클래스를 생성해야 하는데, 그전에 Logger, Appender, Layout 에 대해서 설명한다.

Logger(Category)는 로깅에 대한 요청을 담당하고, 로그를 남기는 역할을 한다. Logger 는 로그 메시지를 Appender 에 전달한다. microlog4android 에서는 FileAppender(파일로 로그 출력), LogCatAppender(LogCat 으로 로그 출력), DatagramAppender(user Datagram Protocol 에 로그를 출력), SyslogAppender (syslog 프로토콜로 로그를 출력) 등의 Appender 들이 있다. 본 문서는 FileAppender 과 LogCatAppender 을 다룬다. layout 은 출력할 로그의 스타일과 내용을 정의한다. 즉, 프로그램에서 기술한 로그 메시지 뿐만 아니라 날짜, 시간, 레벨등의 부가적인 정보도 합쳐서 출력할수 있다.

```
public class Log4Manager
{
  final static private String mFileName = "alarmmon_log";
  final static private Level mDefaultLevel = Level.INFO;
  static private boolean mShowLogCat = true;
  static private String mDefaultPattern = " %d{ISO8601} [%P] %m ";
  static private Logger logger = null;
```

```
    private static Logger getLogger(){
        if (logger == null) {
            logger = LoggerFactory.getLogger();
            logger.setLevel(mDefaultLevel);

            PatternFormatter formatter = new PatternFormatter();
            formatter.setPattern(mDefaultPattern);

            if (mShowLogCat)// write to LogCat{
                LogCatAppender logCatAppender = new
LogCatAppender();
                logCatAppender.setFormatter(formatter);
                logger.addAppender(logCatAppender);
            }
            // write to text file of SD-card
            FileAppender fileAppender = new FileAppender();
            fileAppender.setAppend(true);   // 파일끝에 추가할지 여부
            fileAppender.setFileName(mFileName);

            fileAppender.setFormatter(formatter);
            logger.addAppender(fileAppender);
        }
        return logger;
    }
    ...
}
```

위 그림의 Log4Manager 클래스는 microlog4android 라이브러리를 사용하여 로그를 기록하기 위해 생성한 클래스이다. 로깅을 위해서는 먼저 LoggerFactory 클래스의 getLogger 정적 메소드를 호출하여 Logger 객체를 생성해야한다.

그리고 setLevel 메소드를 통해서 이 Logger 객체의 레벨(level)을 할당해야한다. microlog4android 에서 로그는 다음과 같은 레벨(level)을 가지고 있다.

FATAL	치명적인 에러가 발생하여 시스템이 더 이상 유지될수 없는 상태를 나타낸다.
ERROR	예상치 못한 에러가 발생한 상태를 나타낸다.
WARN	예상치 못한 문제가 발생했지만, 자체적으로 처리가 가능한 상태를 나타낸다.
INFO	어플리케이션 동작과 관련된 일반적인 정보를 나타낸다.
DEBUG	디버깅을 위해 필요한 일반적인 정보를 상세히 나타낸다.
TRACE	가장 상세한 정보를 나타낸다.

기본적으로 Logger 객체는 DEBUG 레벨이 할당되어있다. 로그 레벨의 크기는 FATAL > ERROR > WARN > INFO > DEBUG > TRACE 순서로 구성되며. 어플리케이션 내에서 로그요청이 발생했을 경우, 로그 요청의 레벨이 Logger 객체의 레벨보다 크거나 같을 경우에만 로그가 출력된다. 현재 Log4Manager 클래스는 Logger 객체의 레벨로 INFO 를 설정해주고 있기 때문에, 로그 요청의 레벨이 INFO, WARN, ERROR, FATAL 일 경우에만 로그가 출력된다. Logger 객체의 레벨을 바꾸고 싶다면 mDefaultLevel 의 값을 변경해주면 된다.

Logger 객체를 생성한 이후엔 appender 를 통해서 출력될 메시지의 패턴을 지정하기 위해서 PatternFormatter 객체를 생성하고 setPattern 메소드를 통해서 패턴문자열을 지정해준다. 다음 표와 같은 패턴들을 사용하여 로그 포맷팅을 할수있다.

[표 로그 포맷 형식]

형식	설명
%d	로깅 이벤트가 발생한 시간을 출력한다. %d{yyyy-MMM-dd} 같이 아래 표의 SimpleDateFormat 에 따른 포맷팅을 하면 됨. 그 외 %d{ISO8601} 처럼 쓸 수도 있으며, 이 경우 “2012-12-02 04:01:31,561” 와 같은 형태로 표현됨.
%l	로그가 발생한 위치의 LocationInfo 정보를 출력.
%F	로그가 발생한 파일명을 출력.
%L	로그가 발생한 소스의 행번호를 출력.
%M	로그가 발생한 메소드의 이름을 출력.

%m	로그 메시지를 출력.
%n	개행 문자를 출력.
%p	DEBUG, INFO, WARN, ERROR, FATAL 등의 로그 레벨을 출력
%r	응답 시간을 출력.(ms)
%l	로그를 생성한 쓰레드명을 출력.
%x	NDC 의 값을 출력.

[표 SimpleDateFormat 에 따른 Date 포맷 형식]

형식	설명	보기
G	기원(BC,AD)	AD
y	년	2012
M	월(1~12 월)	7
w	년의 몇 번째 주(1~53)	28
W	월의 몇 번째 주(1~5)	2
D	년의 몇 번째 일(1~366)	194
d	월의 몇 번째 일(1~31)	11
F	월의 몇 번째 요일(1~5)	3
E	요일	Mon
a	오전/오후(AM,PM)	PM
H	시간(0~23)	0
k	시간(1~24)	24
K	시간(0~11)	10
h	시간(1~12)	12
m	분(0~59)	30
s	초(0~59)	55
S	천분의 1 초(0~999)	985
z	Time zone(General Time Zone)	GMT+9:00
Z	Time zone(RFC 822 time zone)	-0800

Logger 객체를 생성하고, 로그 포맷을 설정해 주었다면, 전달될 로그 메시지를 전달할 Appender 를 추가시켜줘야 한다. 만약 로그 메시지를 SD 카드 이외에 안드로이드 DDMS 의 LogCat 을 통해서도 확인하고 싶다면 LogCatAppender 객체를 생성하고 setFormatter 메소드를 통해 로그 포맷을 설정해준뒤 Logger 객체의 addAppender 메소드를 통해서 추가시켜주면 된다. Log4Manager 클래스에서는 mShowLogCat 의 값을 true 로 주면 이와 같은 과정을 수행한다.

SD 카드에 파일 형태로 로그를 기록하기 위해서는 FileAppender 객체를 생성하고 setFormatter 메소드를 통해서 로그 포맷을 설정해준다. 로그를 출력할 때 SD 카드에 저장된 파일 끝에 추가할지 여부는 setAppend 메소드를 통해서 설정해준다. 그리고 나서 setFileName 메소드의 매개변수로 SD 카드에 저장될 파일의 상대주소를 지정해주면 해당 위치에 로그들이 저장된다.

FileAppender 는 지정한 파일에 계속 로그를 출력하게 되면, 그 파일의 크기가 지나치게 커질수 있어 계획적인 로그관리가 힘들어질수 있다. 그렇기 때문에 해당 파일이 일정크기 이상이 되면 기존 파일을 백업파일로 바꾸고 처음부터 다시 로깅을 시작하게 하는 등의 방법이 필요하다. 다음은 SD 카드에 로깅되고 있는 파일이 특정 바이트를 초과하고 있는지 여부를 확인하는 메소드이다.

```
public static boolean isExcess(int bytes)
{
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)||
Environment.MEDIA_MOUNTED_READ_ONLY.equals(state))
{
String mRootPath =
Environment.getExternalStorageDirectory().getAbsolutePath();
File fRingtone = new File(mRootPath + "/" + mFileName);

if (fRingtone.exists())
{
long size = fRingtone.length();
if (size > bytes)
return true;
```

```
        }
        }
        return false;
        }
```

SD 카드에 로깅되고 있는 파일의 이름을 지정할 수 있다는 것은 설정한 날짜 또는 조건에 맞춰 로깅을 수행할수 있다는 것을 의미한다. 예를 들어 로깅파일을 하루단위로 저장하고 싶을경우 파일명을 다음과 같이 생성하면 된다.

```
        Calendar cal = new GregorianCalendar();
        SimpleDateFormat mSimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
        String filename =  mSimpleDateFormat.format(cal.getTime())+".log";
```

이처럼 로깅될 파일명을 설정할 때, 다음과 같은 Date 포맷형식에 맞춰서 파일명을 설정해줄경우, 다양한 조건에 맞춰서 로깅을 수행할수 있다.

Date 포맷	의미
yyyy-MM	로깅 파일을 월 단위로 관리
yyyy-ww	로깅 파일을 주 단위로 관리
yyyy-MM-dd	로깅 파일을 하루 단위로 관리
yyyy-MM-dd-a	로깅 파일을 자정과 정오 단위로 관리
yyyy-MM-dd-HH	로깅 파일을 시간 단위로 관리
yyyy-MM-dd-HH-mm	로깅파일을 분단위로 관리

[별첨3] Logdod의 실제 적용 사례 - 알람몬

"알람몬"은 알람앱이기 때문에 내부적으로 알람이 울릴 때 내부적으로 어떻게 동작했는지에 대한 정보를 microlog4android 라이브러리를 사용하여 SD 카드에 저장하고 있다. 그리고 다음과 같은 화면을 통해서 알람이 동작하지 않았을 경우 사용자가 오류가 발생한 내용에 대해서 메시지를 입력할수 있도록 하고 있다.

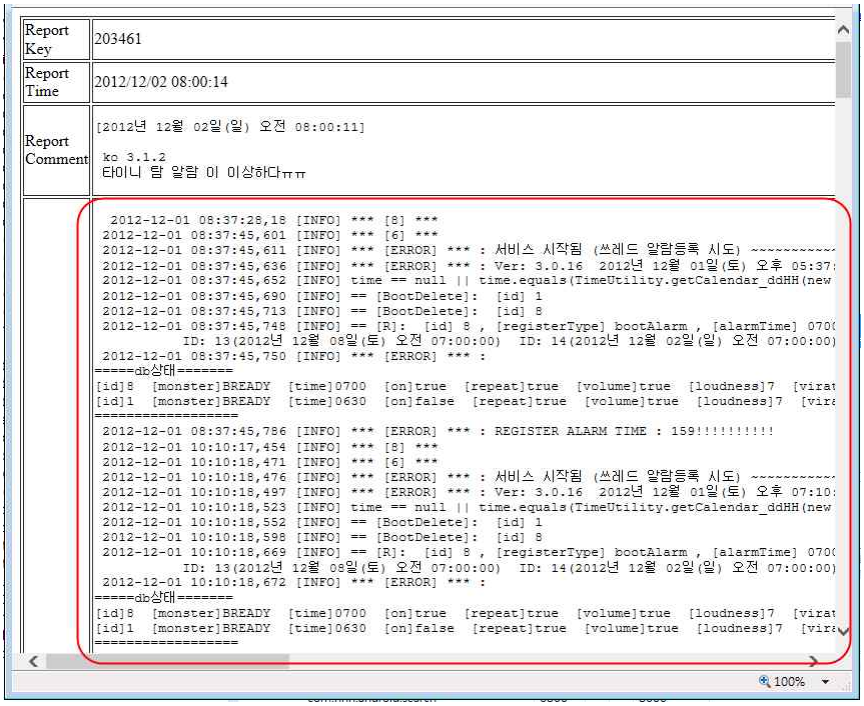
그리고 "못된 오류! 말랑에 알려주기" 버튼을 누르게 되면 사용자가 입력한 메시지와 함께 로깅파일의 내용도 서버로 전송하게 된다. 그리고 다음 그림과 같은 형태로 사용자가 전송한 메시지를 써드파티 서버를 통해서 확인하고 유지보수를 하게 된다.

[그림 4. '알람몬'의 오류보고 화면]

203957	2012/12/02 09:12:38	[2012년 12월 02일(일) 오전 06:38:51] zh-rTW 3.1.2 KKKKKKKKKKKKKKKKK
203853	2012/12/02 09:01:58	[2012년 11월 19일(월) 오전 12:27:50] zh-rTW 3.1.2 HI
203461	2012/12/02 08:00:14	[2012/12/02(周日) 上午 09:03:03] SKY IM-A650S PANTECH SKY 2.2.1 KR ko 3.1.2 타이니 람 알람 이 이상하다ㅠㅠ
203429	2012/12/02 07:57:23	[2012년 12월 02일(일) 오전 07:57:22] LG-F200S LGE lge 4.0.4 KR ko 3.1.2 SKTelecom 소리가 안나요
203428	2012/12/02 07:56:54	[2012년 12월 02일 일요일 오전 07:58:12] korean 2.4.11 잠을 자는데 너시끄럽게울렸어요

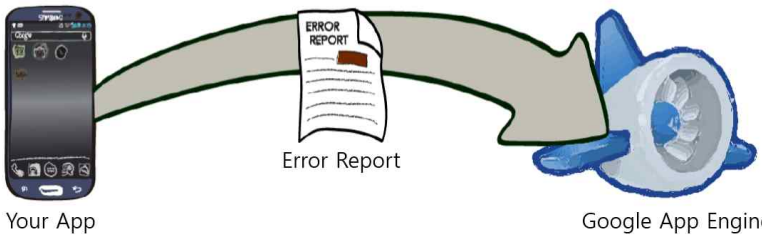
100%

[그림 5.사용자가 보낸 오류보고 리스트]



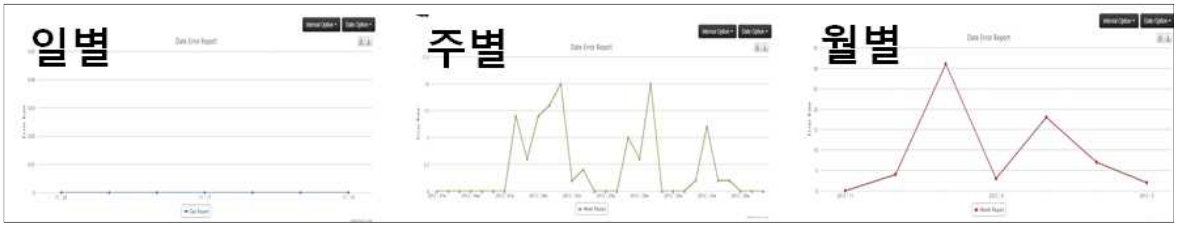
[그림 로깅파일의 내용 확인]

하지만 logdog 의 경우 AppEngine Appender 를 제공하고 있기 때문에, 자동으로 버그리포트를 작성하여 JSON 형태로 AppEngine 으로 전송된다.



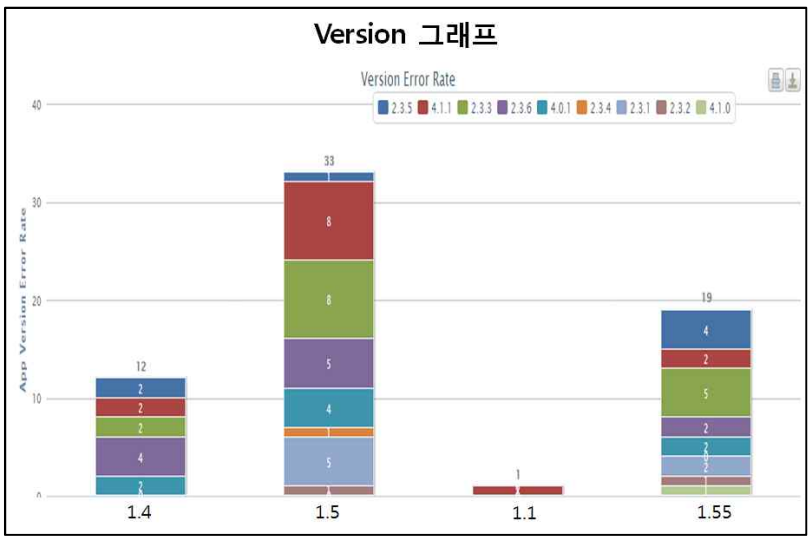
[그림 6. 버그리포팅 기능을 제공하는 AppEngine Appender]

Logdog 은 다음 그림과 같이 웹페이지를 통해서 일별, 주별, 월별 에러 변화량을 알려주는 그래프를 제공하여, 개발자 앱의 에러의 변화 흐름을 보다 정밀하게 체크해볼수 있도록 개선이 이루어졌다.



[그림 7. 일별, 주별, 월별 그래프]

그리고 다음 그림과 같이 App 버전과 OS 버전에 대한 Stack Bar 형태의 그래프를 그려, 이 둘의 상관관계를 보다 확실하게 파악할 수 있도록 개선되었다.



[그림 8. 버전별 에러 분포도]

[별첨4] Funf 예제

가. Tutorial

Wifi Scanner

Funf 를 사용하여 Wifi 를 스캔 하여 정보를 가져오는 어플리케이션을 만드는 예제를 보겠다. 주기적으로 Wifi 엑세스 포인트와 자신의 위치를 검색하는 어플리케이션으로 버튼을 통해서도 정보를 가져올 수 있도록 하고, SD 카드에 결과물을 저장시키는 예제이다.
예제 소스는 git 을 통하여 통하여 받거나 Funf Google Code 에서 받을 수 있다.

https://code.google.com/p/funf-open-sensing-framework.samples/

개요

예제 소스 에서의 MainPipeline 클래스는 ConfiguredPipeline 클래스를 custom 구현한 것이다.
Probe 는 데이터를 내보낼 때, 파이프 라인은 이러한 데이터 요청을 JSON 으로 직렬화하고, 암호화와 저장을 해주는 데이터베이스 서비스로 보낼 수 있다.
ConfiguredPipeline 클래스는 데이터를 요청 했을 때 어떤 일이 벌어지는지, 데이터가 어떻게 되는지 제어 한다.
LauncherReceiver 는 일반적인 시스템 이벤트를 받기 위해 대기하거나 MainPipeline 의 서비스를 시작하는 등의 MainPipeline 서비스를 유지하는 간단한 일을 한다.
JsonUtils 는 Funf 데이터를 JSON 문자열로 직렬화하여 intent 를 통하여 데이터를 전달하도록 하기 위해 Gson 을 사용한다.

Configuring Collection

MainPipeline 는 assets/default_config.json 파일을 사용하여 부트스트랩한다.
어플리케이션이 설치가 되면 이 파일은 읽기 전용이 되며, 이 파일을 통하여 Wi-Fi 와 위치 데이터의 정보를 수집할 범위 등을 설정할 수 있다.

```
{
        "name": "WifiScanner",
    "version":1,
    "dataArchivePeriod":3600,
```

```
        "dataRequests":{
            "edu.mit.media.funf.probe.builtin.LocationProbe": [
                { "PERIOD": 900, "DURATION": 30 }
            ],
            "edu.mit.media.funf.probe.builtin.WifiProbe": [
                { "PERIOD": 900 }
            ]
        }
    }
```

Prove 서비스와 Framework 서비스, 리시버는 AndroidManifest.xml 에서 아래 소스와 같이 추가한다.

```
    <!-- Probe Services -->
    <service
    android:name="edu.mit.media.funf.probe.builtin.LocationProbe"></service>
    <service
    android:name="edu.mit.media.funf.probe.builtin.WifiProbe"></service>

    <!-- Framework services -->
    <service android:name=".MainPipeline"></service>
    <service
    android:name="edu.mit.media.funf.storage.NameValueDatabaseService">
    </service>
    <receiver android:name=".LauncherReceiver" android:enabled="true">
            <intent-filter>
                    <action
    android:name="android.intent.action.BATTERY_CHANGED" />
                    <action
    android:name="android.intent.action.BOOT_COMPLETED" />
                    <action
```

```
android:name="android.intent.action.DOCK_EVENT" />
                <action
android:name="android.intent.action.ACTION_SCREEN_ON" />
                <action
android:name="android.intent.action.USER_PRESENT" />
        </intent-filter>
</receiver>
```

그리고 어플리케이션이 원활하게 데이터를 수집할 수 있도록 배터리 상태, 절전모드 조절 등의 위한 퍼미션을 추가한다.

```
<!-- Launching -->
    <uses-permission
android:name="android.permission.BATTERY_STATS" />

    <!-- All probes -->
    <uses-permission
android:name="android.permission.WAKE_LOCK"/>

    <!-- Location probe -->
        <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <!-- Wifi probe -->
    <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission
android:name="android.permission.CHANGE_WIFI_STATE"/>
```

```
        <!-- Storage -->
        <uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

어플리케이션 실행하기

내부의 세부적인 기능이 구현되어 있는 상태는 아니지만, 우선 실행에 필요한 구성 요소는 모두 갖추고 있다. 이제 내부 기능을 구현하여 요청된 데이터를 수집하고 정기적으로 SD 카드에 저장되도록 할 것이다.

우선 제대로 실행가능한 상태가 됐는지 확인하기 위해 이클립스에서 "Run -> Run As -> Android Application"으로 어플리케이션을 설치한다.
이상 없이 실행이 되면 현재 기기에는 아무런 액티비티도 구현이 안되어 있기 때문에 안드로이드에서 "Settings -> Appplications -> Running Services" 으로 들어가 실행되고 있는 응용프로그램 목록에서 “Wifi Scanner”가 실행되고 있는지를 확인 할 수 있다. 만약 목록에 어플리케이션이 없다면, 전원 버튼으로 화면을 껏다 켜서 파이프라인 서비스를 다시 시작해보자. “Wifi Scanner”가 실행중인 서비스에 있다면, MainPipeline 에 WifiProbe 와 LocationProbe 가 포함되어 있는 것이다. 각 Probe 의 데이터 요청이 있는 한 서비스는 오래 활성화가 된다.

기본 인터페이스

이제 몇 가지의 옵션을 만들어서 사용자가 조금 더 편하게 데이터 수집을 할 수 있도록 할 것 이다. 사용자가 데이터 수집을 활성, 비활성화 할 수 있는 기능을 구현하고, Wi-fi 가 연결되어 있을 경우에는 즉시 위치를 스캐닝하여 위치 데이터를 수집하도록 할 것이다.
메인 엑티비티인 MainActivity.java 을 만들고, AndroidManifest.xml 에 엑티비티를 추가한다.

```
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category   android:name="android.intent.category.LAUNCHER"
/>
        </intent-filter>
    </activity>
```

Next, 다음은, "res/layout/main.xml"파일에서 사용자가 편리하게 조작을 할수 있도록 레이아웃을 만든다. 만들 폼들은 다음과 같다.

앱에 대한 설명이 있는 TextView
데이터 수집을 활성, 비활성 하는 checkbox
모여진 데이터를 SD card 로 이동시키는 button

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    <TextView android:text="Wifi Scanner periodically scans your
location and surrounding wifi access points, and saves that information."
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"/>
    <LinearLayout android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <TextView android:text="Enabled"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"/>
                <CheckBox android:id="@+id/enabledCheckbox"
                        android:layout_width="wrap_content"
                    android:layout_height="wrap_content"/>
        </LinearLayout>
        <Button android:id="@+id/archiveButton"
                        android:text="Save Data to SD Card"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"/>
```

```
</LinearLayout>
```

이젠 onCreate 함수를 오버라이드하여 MainActivity 에서 각 폼들에 대한 이벤트에 응답할 수 있도록 한다. 버튼을 클릭하면 MainPipeline 에 활성, 비 활성화 할것인지, 수집된 데이터를 SD Card 로 보낼것인지에 대한 인텐트를 보내게 된다.

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

 final Context context = this;

 CheckBox                                          enabledCheckbox
 =(CheckBox)findViewById(R.id.enabledCheckbox);
 enabledCheckbox.setChecked(MainPipeline.isEnabled(context));
 enabledCheckbox.setOnCheckedChangeListener(new
        OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
        Intent archiveIntent = new Intent(context, MainPipeline.class);
        String action = isChecked ? MainPipeline.ACTION_ENABLE :
MainPipeline.ACTION_DISABLE;
        archiveIntent.setAction(action);
        startService(archiveIntent);
        }
 });

 Button archiveButton = (Button)findViewById(R.id.archiveButton);
 archiveButton.setOnClickListener(new OnClickListener() {
        @Override
```

```
            public void onClick(View v) {
            Intent archiveIntent = new Intent(context, MainPipeline.class);
            archiveIntent.setAction(MainPipeline.ACTION_ARCHIVE_DATA);
            startService(archiveIntent);
            }
    });
}
```

Live Interaction

위 인터페이스는 사용자가 데이터 수집을 시작, 정지할 수 있지만, 데이터가 실제로 수집되고 있는지에 대한 피드백이 전혀 없다. 지금부터는 데이터가 모인 개수를 파악하여 화면에 보여주는 기능을 추가 할 것이다

data Count

우선 Main Pipeline 에 다음 코드를 추가하여 카운트를 증가시키는 함수를 만든다

```
public static final String SCAN_COUNT_KEY = "SCAN_COUNT";
public static long getScanCount(Context context) {
  return getSystemPrefs(context).getLong(SCAN_COUNT_KEY, 0L);
  }
private void incrementCount() {
  boolean success = false;
  while(!success) {
            SharedPreferences.Editor editor = getSystemPrefs().edit();
            editor.putLong(SCAN_COUNT_KEY, getScanCount(this) + 1L);
            success = editor.commit();
  }
}
```

그 다음 오버라이딩된 onDataReceived 메소드에서 데이터를 수신할 때 incrementCount 메소드를 호출하여 카운트를 하나 증가시키도록 한다.

```
@Override
public void onDataReceived(Bundle data) {
        super.onDataReceived(data);
        incrementCount();
}
```

그럼 이제는 카운트를 알 수 있는 변수가 있으므로 main.xml 에 TextView 를 만들어 엑티비티에 카운트가 보일수 있도록 한다.

```
<TextView android:id="@+id/dataCountText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

마지막으로 TextView 에 카운트된 값을 설정하고 TextView 와 SharedPreferences 클래스에 실시간으로 업데이트 되고 있는지 확인 하기 위해 MainActivity 클래스에 아래 메소드들을 추가한다.

```
@Override
public void onSharedPreferenceChanged(SharedPreferences
sharedPreferences, String key) {
        Log.i("WifiScanner", "SharedPref change: " + key);
        if (MainPipeline.SCAN_COUNT_KEY.equals(key)) {
                updateScanCount();
        }
}

private void updateScanCount() {
        TextView dataCountView =
(TextView)findViewById(R.id.dataCountText);
        dataCountView.setText("Data Count: " +
MainPipeline.getScanCount(this));
}
```

onCreate 메소드에 추가한다.

```
MainPipeline.getSystemPrefs(this).registerOnSharedPreferenceChangeLi
stener(this);
updateScanCount();
```

Scan Now Button

사용자가 특정한 순간에 즉시 스캔을 할 필요를 대비하여 버튼을 구현한다. main.xml 에 Button 을 추가한다.

```
<Button android:id="@+id/scanNowButton"
                android:text="Scan Now"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
```

그 다음 MainActivity 내부 onCreate 메소드에 버튼 클릭 리스너를 구현하여 내부에 즉시 스캔하는 기능을 구현한다.

```
Button scanNowButton = (Button)findViewById(R.id.scanNowButton);
scanNowButton.setOnClickListener(new OnClickListener() {
 @Override
 public void onClick(View v) {
        Intent runOnceIntent = new Intent(context, MainPipeline.class);
        runOnceIntent.setAction(MainPipeline.ACTION_RUN_ONCE);

 runOnceIntent.putExtra(MainPipeline.RUN_ONCE_PROBE_NAME,
WifiProbe.class.getName());
        startService(runOnceIntent);

 runOnceIntent.putExtra(MainPipeline.RUN_ONCE_PROBE_NAME,
```

```
        LocationProbe.class.getName());
                startService(runOnceIntent);
          }
        });
```

Getting Data

Funf 에 의해 수집된 데이터는 주기적으로 SQLite 데이터베이스 파일로 SD 카드에 저장된다. 파일이 저장되는 위치는 아래와 같다.

"/sdcard/edu.mit.media.funf.wifiscanner/main".

Data analysis

암호화 되어 있는 데이터베이스 파일은 Funf 스크립트 패키지를 활용하여 일반적인 데이터베이스 파일로 변환할 수가 있다.
일반적인 사용법은 아래와 같다.

```
        > python dbdecrypt.py *.db
        ... enter password (changeme by default)
        > python dbmerge.py *.db
```

생성된 데이터 베이스 파일은 SQLite command line program, SQLite Database Browser 등의 프로그램을 통해서 내용을 확인 할 수 있다.

[별첨5] Android Billing Library사용 예제

안드로이드 앱 개발시 in-app billing 을 직접 구현하는 것은 앞절에서 본것과 같이 많은 부분에 대해서 고려해줘야 하기 때문에 꽤나 번거롭고 많은 시간을 들여야 한다. 이에 본 문서에서는 Android Billing Library 를 이용하여 어떻게 다룰수 있는지 살펴보겠다.
다음 코드는 Android Billing Library 를 어떻게 쓰는지 보여주는 간단한 예이다.

```
public class Dungeons extends Activity {
…
private static final int DIALOG_BILLING_NOT_SUPPORTED_ID = 2;
private AbstractBillingObserver mBillingObserver;
private CatalogEntry mSelectedItem;

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
mBillingObserver = new AbstractBillingObserver(this) {
public void onBillingChecked(boolean supported) {
Dungeons.this.onBillingChecked(supported);
}
public void onPurchaseStateChanged(String itemId, PurchaseState state) {
Dungeons.this.onPurchaseStateChanged(itemId, state);
}
public void onRequestPurchaseResponse(String itemId,
ResponseCode response){
Dungeons.this.onRequestPurchaseResponse(itemId, response);
}
public void onSubscriptionChecked(boolean supported) {
Dungeons.this.onSubscriptionChecked(supported);
}
};

setContentView(R.layout.main);
```

```
setupWidgets();
BillingController.registerObserver(mBillingObserver);
BillingController.checkBillingSupported(this);
BillingController.checkSubscriptionSupported(this);
updateOwnedItems();
}

@Override
protected void onDestroy() {
BillingController.unregisterObserver(mBillingObserver);
super.onDestroy();
}

public void onBillingChecked(boolean supported) {
if (supported) {
restoreTransactions();
…
} else {
…
}
}

public void onPurchaseStateChanged(String itemId, PurchaseState state) {
updateOwnedItems();
}

public void onRequestPurchaseResponse(String itemId, ResponseCode
response) {

}

public void onSubscriptionChecked(boolean supported) {

}
```

```
/**
 * Restores previous transactions, if any. This happens if the application
 * has just been installed or the user wiped data. We do not want to do this
 * on every startup, rather, we want to do only when the database needs to
 * be initialized.
 */
private void restoreTransactions() {
if (!mBillingObserver.isTransactionsRestored()) {
BillingController.restoreTransactions(this);
}

private void setupWidgets() {
      …
    if (mSelectedItem.managed != Managed.SUBSCRIPTION) {
   BillingController.requestPurchase(Dungeons.this, mSelectedItem.sku,
true /* confirm */, null);
} else {
BillingController.requestSubscription(Dungeons.this, mSelectedItem.sku,
true /* confirm */, null);
}
      …
}

private void updateOwnedItems() {
List<Transaction> transactions = BillingController.getTransactions(this);
final ArrayList<String> ownedItems = new ArrayList<String>();
for (Transaction t : transactions) {
if (t.purchaseState == PurchaseState.PURCHASED) {
ownedItems.add(t.productId);
}
}
      …
}
```

```
    }
```

Google Play Application 이 in-app billing 을 지원하는지, 어떤 API 버전을 지원하는지 확인 하기 위해서 BillingController 클래스의 정적 메소드인 onBillingChecked(Context context)을 호출하면 onBillingChecked(boolean supported) 메소드가 호출되면서 지원 여부를 확인할 수 있다.

Subscriptions 구매유형을 지원하는지 확인하기 위해서 BillingController 클래스의 정적 메소드인 checkSubscriptionSupported(Context context) 를 호출 하면 onSubscriptionChecked(boolean supported) 메소드가 호출되면서 지원 여부를 확인할수 있다.

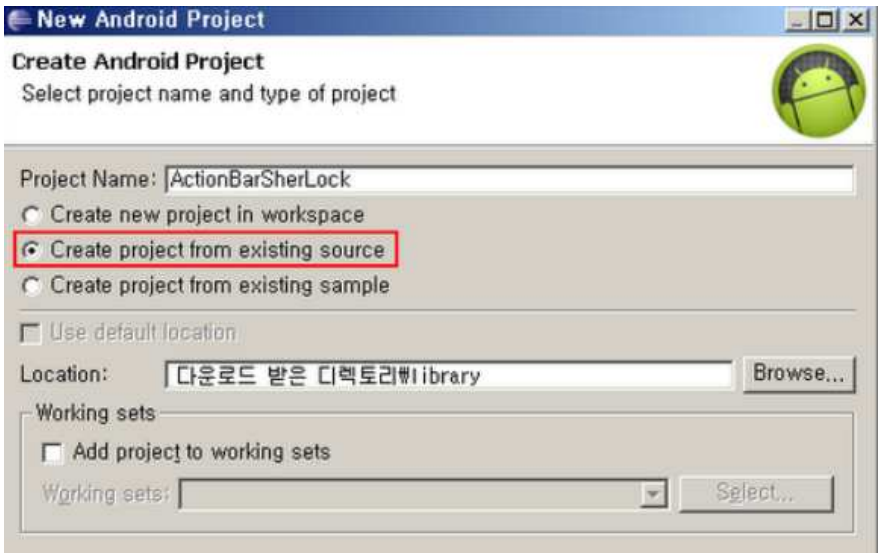
In-app products 상품에 대한 구매 요청시에는 싶다면 BillingController 클래스의 정적 메소드인 requestPurchase(Context context, String itemId, boolean confirm, String developerPayload) 를 호출하고, Subscriptions 상품에 대한 구매 요청시에는 BillingController 클래스의 정적 메소드인 requestSubscription(Context context, String itemId) 을 호출한다. 그리고 구매정보를 복원하기 위해서는 BillingController 클래스의 정적 메소드인 restoreTransactions(Context context)을 호출한다. 그리고 구매 요청 후 구매 상태의 변화가 생겼거나 구매정보 복원 요청을 했을경우 onPurchaseStateChanged(String itemId, PurchaseState state) 메소드가 호출되고 BillingController 클래스의 정적 메소드인 getTransactions(Context context)을 호출하면 모든 거래 정보를 받을수 있다.

기존 Android SDK 에서 제공하고 있는 in-app billing sample Application 의 경우 Android Market Application 에 여러가지 요청을 할 때 일일이 key-value 쌍들을 설정해줘야 했고, 거래정보가 담겨있는 JSON 문자열일 일일이 파싱해서 분석해야 했고, Android Market Application 이 보내준 IN_APP_NOTIFY 에 대해서 CONFIRM_NOTIFICATIONS 응답을 보내줘야 했고, Android Market Application 이 보내준 응답에 대한 무결성 검사를 일일이 해줘야 했다. 하지만 InAppBillingLibrary 의 경우 이런 과정들은 내부적으로 다 처리하고 간단하게 해당 과정에 대한 메소드만 호출해주기 때문에 in-app billing 기능을 안드로이드 앱에 탑재할 때 보다 생산성이 높고 개발하기 용이함을 확인할수 있다. 다음 그림은 기존 Android SDK 에서 제공하는 in-app billing 샘플 앱과 AndroidBillingLibrary 가 적용된 안드로이드 앱의 구조를 보인다. BillingService 와 Android Market Application 간의 메시지를 송수신하는 로직을

AndroidBillingLibrary 에서 모두 수행하기 때문에 개발자가 BillingService 와 Android Market Application 간의 메시지를 송수신 과정을 잘 몰라도 쉽게 in-app Purchase 를 적용할수 있을 정도로 구조가 보다 간결해졌음을 확인할 수 있다.

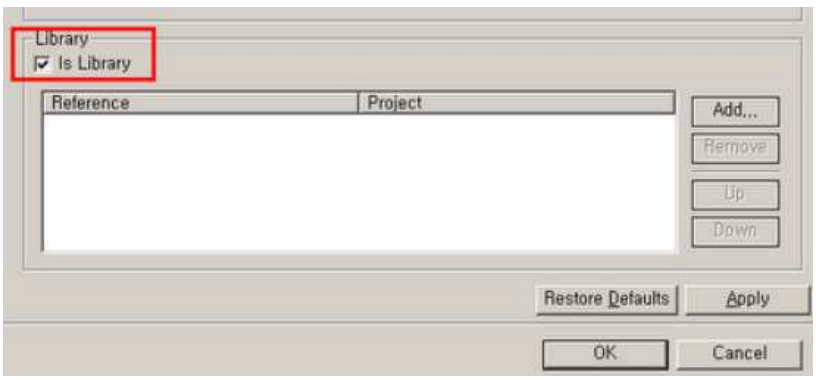
[별첨6] 액션바셜록 라이브러리 프로젝트 생성하기

이클립스(Eclipse)를 실행 시킨 후 File > New > Project > Android Project 를 선택한다. 아래그림처럼 "Create project from existing source"를 선택 후 액션바셜록 압축을 풀어놓은 디렉토리의 library 디렉토리를 선택한다.



[그림 9. 액션바셜록 라이브러리 생성]

다음단계에서 하단의 Library 항목의 "Is Library"를 체크해준다.



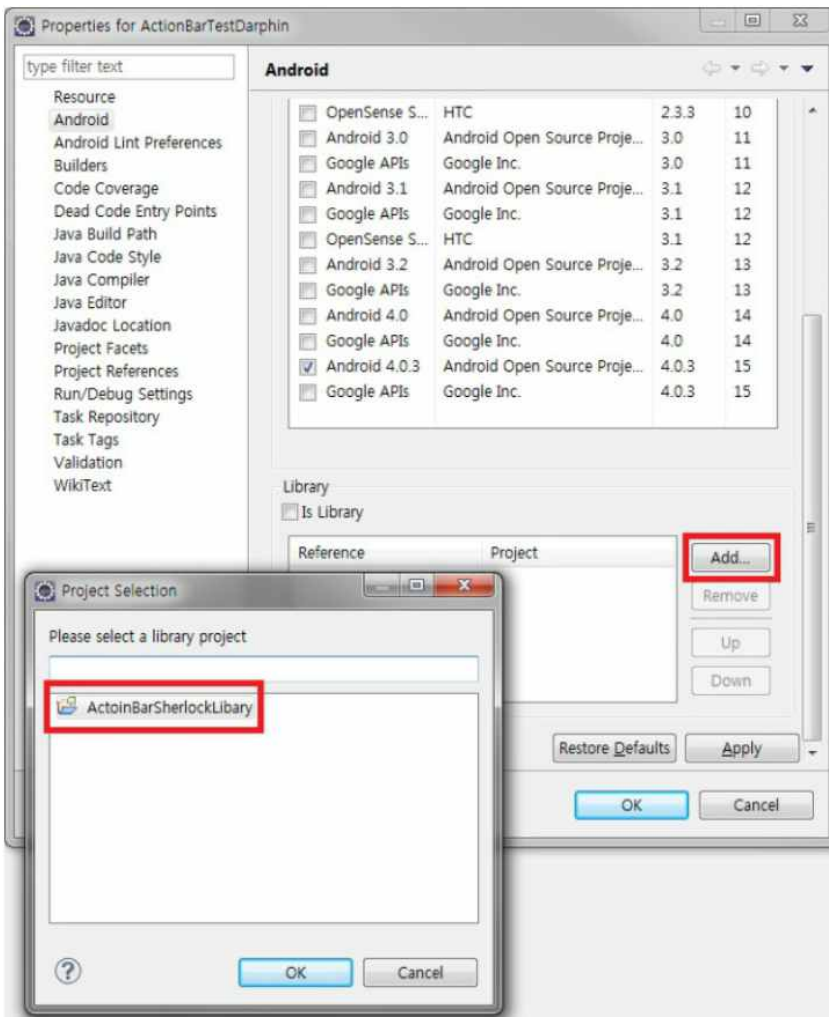
[그림 10. 프로젝트 프로퍼티창]

또한 만약 메이븐(Maven)을 사용한다면 아래의 디펜던시(dependency)를 추가함으로써 간단히 라이브러리를 추가할 수 있다.

```
<dependency>
  <groupId>com.actionbarsherlock</groupId>
  <artifactId>actionbarsherlock</artifactId>
  <version>4.2.0</version>
  <type>apklib</type>
</dependency>
```

액션바셜록 적용하기

액션바셜록을 적용하려는 프로젝트의 속성창으로 이동하여 액션바셜록 라이브러리 프로젝트를 추가해준다. 위 과정을 거치면 기존의 프로젝트나 새로운 프로젝트에 액션바셜록을 사용할 준비가 끝난다.



[그림 11. 프로젝트 프로퍼티창]

그림. 프로젝트에 액션바 셜록 라이브러리 추가

액션바셜록은 API level 11 이하에서 액션바를 사용하기 위해 추가한 라이브러리이므로 API level 11 이상으로 만들어진 프로젝트의 경우는 Actionbar 클래스가 두개 생기게 된다. 기존에 존재하는 Android.app.ActionBar 와 액션바셜록 라이브러리를 추가함으로써 생긴 com.actionbarsherlock.app.ActionBar 가 있다. 하위 버전에서도 액션바를 적용하기 위해선 기존의 Android.app.ActionBar 가 아닌 com.actionbarsherlock.appActionBar 를 임포트(import) 해야한다. 또한 액티비티를 상속받을 때에도 Activity 가 아닌 ShelockActivity 를 상속받아 사용해야 하며, ActionBar 객체를 받아오기 위해서는 기존의 getActionBar() 메소드가 아닌 getSupportActionBar() 메소드를 이용해야한다. 테마의 경우 액션바 기본테마는 android:Theme.Holo 이지만 액션바셜록의 기본테마는 Theme.Sherlock 이며 동일한 UI 를 가진다. 아래 소스코드는 액션바셜록을 적용하지 않고 기본 제공 ActionBar 클래스를 사용하여 액션바를 구현한 코드이다.

```
package com.example.test;

import android.app.ActionBar;
import android.app.ActionBar.Tab;
import android.app.ActionBar.TabListener;
import android.app.Activity;
import android.app.FragmentTransaction;
import android.os.Bundle;

public class MainActivity extends Activity {

  private ActionBar aBar;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);

        aBar = getActionBar();
        aBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        aBar.setTitle("Action Bar");

        ActionBar.Tab tab1 =
aBar.newTab().setText("버튼 1").setTabListener(onTabListener);
        ActionBar.Tab tab2 =
aBar.newTab().setText("버튼 2").setTabListener(onTabListener);
        ActionBar.Tab tab3 =
aBar.newTab().setText("버튼 3").setTabListener(onTabListener);

        aBar.addTab(tab1);
        aBar.addTab(tab2);
        aBar.addTab(tab3);
    }

    TabListener onTabListener = new TabListener() {

        @Override
        public void onTabReselected(Tab tab, FragmentTransaction ft) {
            // TODO Auto-generated method stub

        }

        @Override
        public void onTabSelected(Tab tab, FragmentTransaction ft) {
            // TODO Auto-generated method stub

        }
```

```
        @Override
        public void onTabUnselected(Tab tab, FragmentTransaction ft) {
                // TODO Auto-generated method stub

        }
    };
```

[그림 12.기본 제공 액션바클래스를 이용한 액션바 구현]

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.test"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="11" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/android:Theme.Holo" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
        </application>

    </manifest>
```

[그림 13. 기본 제공 액션바를 이용한 액션바 구현(AndroidManifest.xml)]

아래 소스코드는 액션바셜록 라이브러리에서 제공하는 ActionBar 클래스를 이용하여 액션바를 구현한 코드이다.

```
package com.example.test;

import android.support.v4.app.FragmentTransaction;
import com.actionbarsherlock.app.ActionBar;
import com.actionbarsherlock.app.ActionBar.Tab;
import com.actionbarsherlock.app.ActionBar.TabListener;
import com.actionbarsherlock.app.SherlockActivity;
import android.os.Bundle;

public class MainActivity extends SherlockActivity {

  private ActionBar aBar;

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

        aBar = getSupportActionBar();
        aBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        aBar.setTitle("Action Bar");

        ActionBar.Tab                    tab1                    =
```

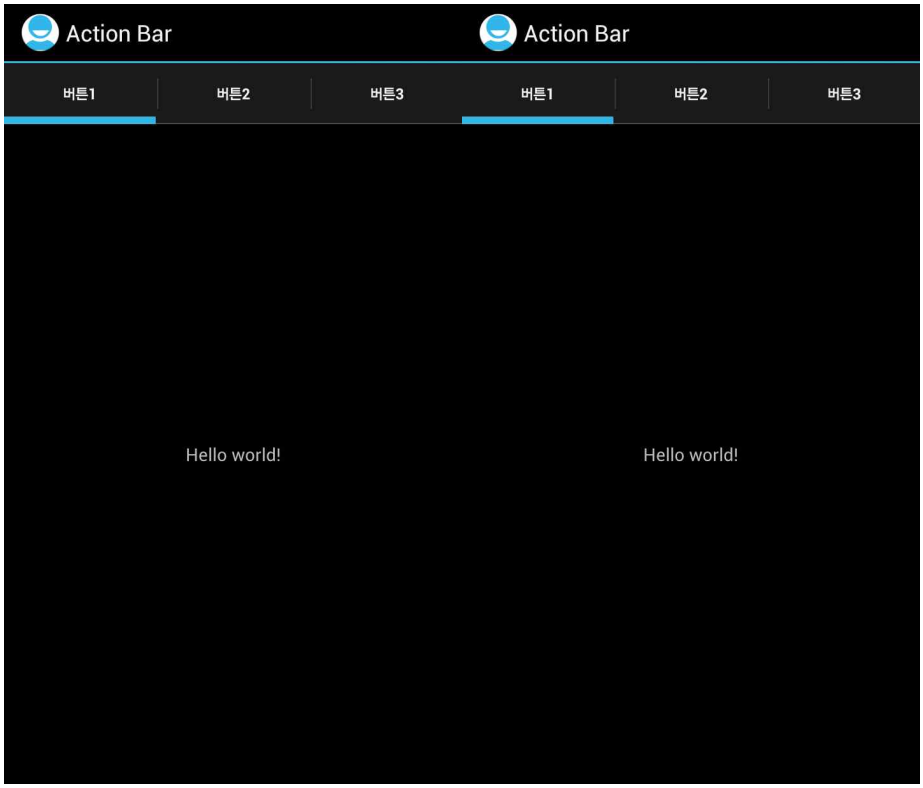
```
aBar.newTab().setText("버튼 1").setTabListener(onTabListener);
        ActionBar.Tab                    tab2                    =
aBar.newTab().setText("버튼 2").setTabListener(onTabListener);
        ActionBar.Tab                    tab3                    =
aBar.newTab().setText("버튼 3").setTabListener(onTabListener);

        aBar.addTab(tab1);
        aBar.addTab(tab2);
        aBar.addTab(tab3);
    }

    TabListener onTabListener = new TabListener() {

        @Override
        public void onTabReselected(Tab tab, FragmentTransaction ft) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onTabSelected(Tab tab, FragmentTransaction ft) {
                // TODO Auto-generated method stub

        }

        @Override
        public void onTabUnselected(Tab tab, FragmentTransaction ft) {
                // TODO Auto-generated method stub

        }
    };
```

[그림 14. 액션바셜록 라이브러리를 이용한 액션바 구현]

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.test"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="11" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.Sherlock" >
        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

[그림 15. 액션바셜록 라이브러리를 이용한 액션바 구현(AndroidManifest.xml)]

위의 두 코드처럼 기본 액션바를 이용하거나 액션바셜록 라이브러리를 이용해 액션바를 구현할 수 있으며 결과화면은 아래와 같이 동일하다.



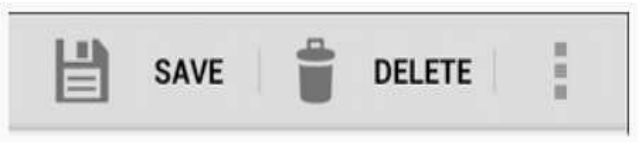
[그림 16. 액션바셜록을 적용한 안드로이드 버전 3.0 이상 레이아웃과 하위버전의 레이아웃]

결국 액션바셜록을 이용하면 안드로이드 버전 3.0 이하에서도 액션바를 이용할 수 있으며 동일한 UI를 얻을 수 있다.

[별첨7] 액션바 컴포넌트들(Actionbar components)

액션아이템(Action Item)

액션바에서는 메뉴버튼을 누르지 않고 바로 메뉴에 접근할 수 있도록 액션아이템(Action item)을 포함하고 있다. 액션아이템은 아이콘과 텍스트 타이틀로 구성되어 있으며 액션아이템으로 나타나지 않는 메뉴들은 오버플로우메뉴(overflow menu)에 존재한다. 아래의 그림은 두 개의 액션아이템과 하나의 오버플로우메뉴버튼을 가진 액션바이다.



[그림 17. 두 개의 액션아이템과 오버플로우메뉴버튼]

액티비티가 처음 시작되면 onCreateOptionsMenu() 호출을 통해 액션바의 메뉴를 생성하며 아래의 예시에서 보이는 것처럼 main_activity.xml 에 정의되어있는 메뉴아이템을 기반으로 만들어진다.

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_activity, menu);
        return true;
    }
```

[그림 18. onCreateOptionMenu 메소드]

XML 파일에서 메뉴아이템을 선언할 때 <item> 엘리먼트에서 android:showAsAction="ifRoom" 설정을 하게되면 액션바에 자리가 있을 경우 액션아이템으로 나타나게 된다. android:showAsAction 속성으로는 아래의 항목들을 지정할 수 있다.

- ifRoom : 액션아이템을 표시할 수 있는 공간이 있다면 액션아이템을 표시
- never : 항상 액션아이템으로 표시하지 않음(기본값)
- withText : 메뉴항목의 아이콘과 텍스트를 함께 액션아이템으로 표시

▸always : 항상 액션아이템으로 표시

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
          android:icon="@drawable/ic_menu_save"
          android:title="@string/menu_save"
          android:showAsAction="ifRoom|withText" />
</menu>
```

[그림 19. main_activity.xml 의 메뉴 정의]

메뉴항목이나 액션항목을 선택했을 때 수행할 작업 구현은 기존과 동일하게 onOptionsItemSelected() 메소드에서 처리한다. 액션바 좌측에 표시되는 아이콘을 눌렀을 때 처리할 동작 또한 동일한 메소드에서 처리할 수 있다. 애플리케이션 아이콘의 id 는 android.R.id.home 으로 onOptionSelected()에서 이 id 를 사용하여 적절한 작업을 수행하도록 구현할 수 있다. 아래의 코드는 두 개의 액션아이템과 하나의 오버플로우메뉴를 가지는 액티비티를 나타낸다. xml 소스를 보면 첫 번째 액션아이템은 공간이 있다면 텍스트와함께 나타나며 두 번째 아이템은 항상 아이콘만 나타나고 세 번째 아이템은 오버플로우메뉴로 나타나게 된다.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:icon="@drawable/ic_menu_add"
          android:showAsAction="ifRoom|collapseActionView"
          android:title="Add"
     />
     <item android:id="@+id/item2"
          android:icon="@drawable/ic_menu_delete"
          android:showAsAction="always"
          android:title="Action item with icon"
     />
```

```
        <item android:id="@+id/item3"
              android:title="Other item"
         />
    </menu>
```

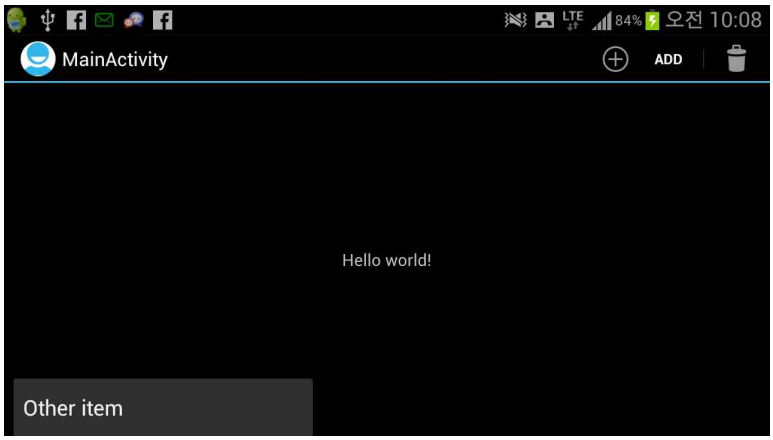
[그림 20. menu.xml]

아래의 코드(MainActivity.java)에서 볼 수 있듯이 위에서 정의된 xml 을 기반으로 액션바를 생성하게된다. onOptionsItemSelected 에서는 메뉴항목 중 애플리케이션 아이콘을 선택했을 때 수행할 동작을 구현하였다.

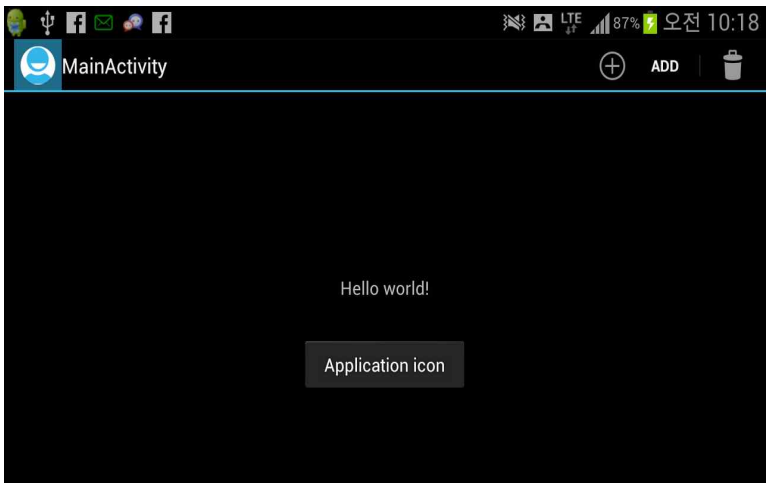
```
    @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.menu, menu);
            return true;
        }

        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            String text = null;

            switch(item.getItemId()){
            case android.R.id.home:
                    text = "Application icon";
                    break;

            default:
                    return false;
            }
            Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
            return true;
        }
```

[그림 21. MainActivity.java]

아래는 MainActivity 의 실행화면으로 두 개의 액션아이템과 하나의 오버플로우메뉴를 볼 수 있다. 오버플로우메뉴는 단말기의 메뉴버튼 유무에 따라 달라지는데 단말기에 메뉴버튼이 없을 경우는(Galaxy Nexus) 액션바 우측에 정상적으로 오버플로우버튼이 생기며 메뉴버튼이 있는 경우는(Galaxy S3) 메뉴버튼을 눌렀을 때 아래 그림처럼 메뉴가 디스플레이된다.



[그림 22.MainActivity 실행화면(Landscape, Galaxy S3)]



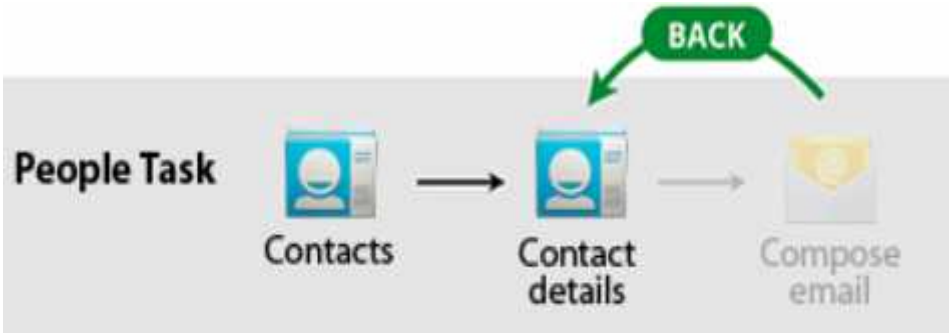
[그림 23. 애플리케이션 아이콘을 선택했을 때 화면(Landscape, Galaxy S3)]

위 그림처럼 액션바 좌측의 애플리케이션 아이콘도 하나의 액션아이템으로 사용가능하다. 일반적으로 애플리케이션 아이콘은 홈 액티비티로 가거나 네비게이팅업(Navigating up) 기능을 가지게된다. 홈 액티비티로 가는 액션의 경우는 Intent.FLAG_ACTIVITY_CLEAR_TOP 플래그를 포함해야한다. 네비게이팅업의 경우는

아래의 그림처럼 아이콘에 화살표가 존재하며 아이콘을 선택하면 윗 단계로 돌아가는데 이는 뒤로가기(Back)버튼과는 다르다.

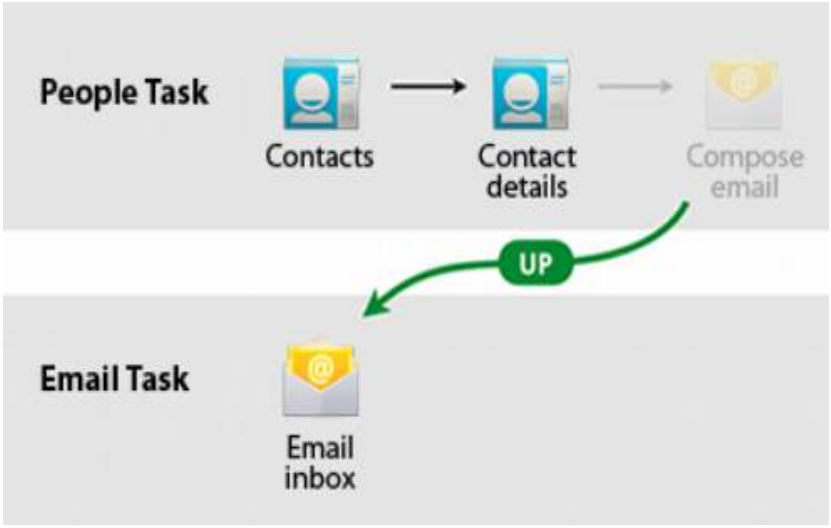
[그림 24. 일반아이콘과 네비게이팅업 아이콘]

주소록에서 한명을 선택해서 메일을 보내는 경우 아래의 그림과 같은 테스크 플로우를 가지는데 이메일 작성 액티비티에서 뒤로가기 버튼을 누를 경우 아래처럼 다시 주소록으로 돌아가게된다.



[그림 25. 주소록에서 이메일 앱 실행 후 뒤로가기 버튼을 누를경우]

하지만 네비게이팅업의 경우는 아래그림처럼 해당 액티비티의 구조적으로 한 단계 위의 액티비티가 실행된다. 이 같이 사용자가 현재 액티비티에서 다른 애플리케이션으로 진입하는 경우에는 Intent.FLAG_ACTIVITY_NEW_TASK 플래그를 추가해야한다.



[그림 26. 주소록에서 이메일 앱 실행 후 네비게이팅 업 버튼을 누른경우]

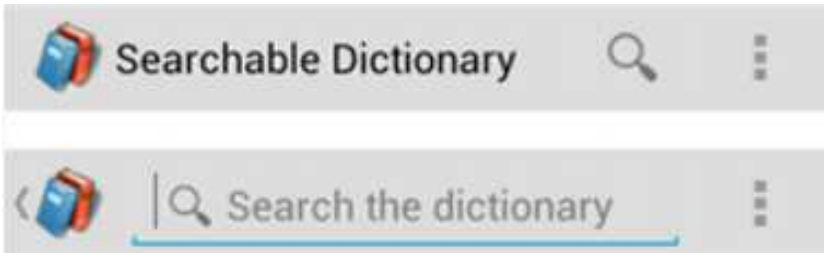
액션바 아이콘을 네비게이팅업 형식으로 표시하기위해선 setDisplayHomeAsUpEnabled(true)메소드를 호출해주어야 하며 액션바의 인스턴스는 getActionBar()메소드로 얻을 수 있다.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        ActionBar actionBar = getActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
        …
    }
```

그림 27. 액션바 아이콘을 네비게이팅업 형식으로 설정

액션뷰(ActionView)

액션뷰는 액션아이템 버튼에 대한 액션으로 액션바에 나타나는 위젯이다. 예를 들면 아래 그림처럼 “Search”라는 액션아이템에 SearchView 액션뷰를 추가할 수 있다. “Search” 액션아이템을 선택한 경우 새로운창이 뜨거나 다이얼로그가 뜨는 것이 아니라 액션바에서 SearchView 가 나타나는 형태이다.



[그림 28. SearchView 액션뷰를 추가한 액션아이템, 위(선택 전) - 아래(선택 후)]

액션뷰를 구현하기 위해선 아래의 소스처럼 menu.xml 리소스에서 android:actionLayout 속성이나 android:actionViewClass 속성을 통해 원하는 위젯을 선언해야한다.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_search"
          android:title="@string/menu_search"
          android:icon="@drawable/ic_menu_search"
          android:showAsAction="ifRoom|collapseActionView"
          android:actionViewClass="android.widget.SearchView" />
</menu>
```

[그림 29. 액션뷰 사용을 위한 android:actionViewClass 속성 설정]

만약 액션뷰에 특정 이벤트를 추가하고 싶다면 onCreateOptionsMenu() 콜백메소드에서 정의할 수 있다. 아래 소스코드처럼 메뉴아이템의 아이디, findItem(), getActionView() 메소드를 이용하면 특정 액션뷰 인스턴스를 구하여 조작할 수 있다.

```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
```

```
        getMenuInflater().inflate(R.menu.options, menu);
        SearchView searchView = (SearchView)
menu.findItem(R.id.menu_search).getActionView();
        // Configure the search info and add any event listeners
        ...
        return super.onCreateOptionsMenu(menu);
    }
```

[그림 30. 특정 액션뷰 인스턴스 얻어오기]

액션뷰는 액티비티의 전환없이 빠르고 풍부한 액션을 사용할 수 있게 해주는데 조그만 화면에서 액션바의 공간을 보존하기 어려운 경우가 생긴다. 이 때 사용할 수 있는 것이 위의 xml 소스코드에서 보이는 android:showAsAction 속성의 collapseActionView 이다. 아래의 그림은 “Search” 액션아이템에 collapseActionView 속성을 준 경우로 선택했을 경우 액션아이템이 없어지고 액션뷰로 확장되는 것을 확인할 수 있다.

[그림 31. showAsAction 속성에 collapseActionView 를 주었을 경우]

만약 액션뷰의 확장여부에 따라 액티비티를 업데이트하고 싶다면 아래의 소스코드처럼 setOnActionExpandListener()를 이용해 콜백으로 받아서 처리할 수 있다.

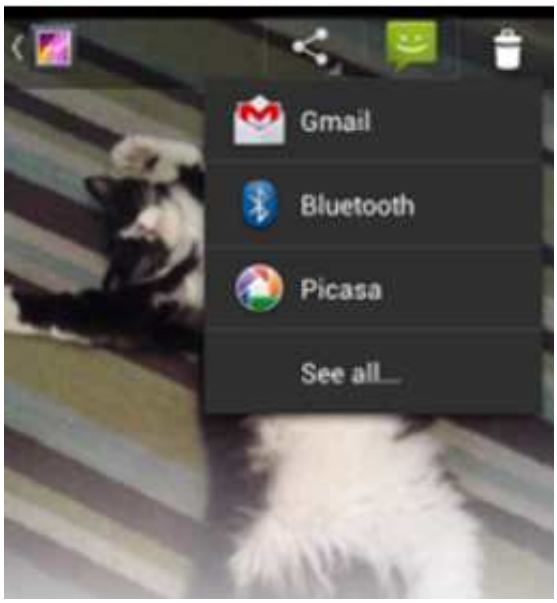
```
@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.options, menu);
        MenuItem menuItem = menu.findItem(R.id.actionItem);
        ...

        menuItem.setOnActionExpandListener(new
OnActionExpandListener() {
```

```
                @Override
                public boolean onMenuItemActionCollapse(MenuItem item)
    {
                    // Do something when collapsed
                    return true;  // Return true to collapse action view
                }

                @Override
                public boolean onMenuItemActionExpand(MenuItem item) {
                    // Do something when expanded
                    return true;  // Return true to expand action view
                }
            });
        }
```

[그림 32. setOnActionExpandListener 를 이용한 액]션뷰 확장에따른 이벤트처리

액션프로바이더(Action Provider)

액션프로바이더(Action Provider)는 액션뷰와 비슷하게 액션아이템을 대체하는 커스텀된 레이아웃으로 아이템의 여러가지 이벤트(액션)를 선택할 수 있다. 액션바에 액션프로바이더를 선언하면 아래의 그림처럼 오버플로우 메뉴형식으로 여러 액션들을 선택할 수 있게 해준다.



[그림 33 . 갤러리앱에 정의된 액션프로바이더(ShareActionProvider)]

일반적으로 액션프로바이더를 상속한 쉐어액션프로바이더(ShareActionProvider)가 많이 쓰이며 위 그림처럼 어떤 방식으로 쉐어 할 것인지를 액션바에서 선택할 수 있다. 일반적인 방식은 다이얼로그를 통해 ACTION_SEND 인텐트를 처리하였으나 쉐어액션프로바이더는 액션바에서 드롭다운 리스트(drop-down list) 방식으로 ACTION_SEND 인텐트를 처리한다. 액션프로바이더를 선언하기 위해선 메뉴를 정의하는 xml 파일에서 android:actionProviderClass 속성을 <item>엘리먼트에 정의해야한다.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_share"
          android:title="@string/share"
          android:showAsAction="ifRoom"

android:actionProviderClass="android.widget.ShareActionProvider" />
    ...
</menu>
```

[그림 34. menu.xml 에 정의한 액션프로바이더가 포함된 액션바]

액션프로바이더를 선택했을 때 나오는 드롭다운리스트는 사용빈도에 기반하여 나오지만 setShareHistoryFileName()과 커스텀 xml 파일(e.g. custom_share_history.xml)을 이용해서 원하는 히스토리로 출력할 수 있다.

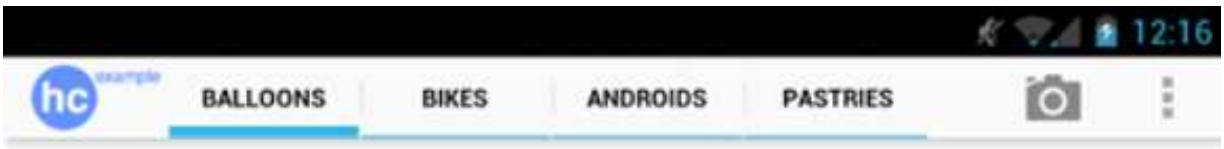
```
private ShareActionProvider mShareActionProvider;
...

@Override
public boolean onCreateOptionsMenu(Menu menu) {
mShareActionProvider = (ShareActionProvider)
menu.findItem(R.id.menu_share).getActionProvider();

    // If you use more than one ShareActionProvider, each for a different
action,
    // use the following line to specify a unique history file for each one.

mShareActionProvider.setShareHistoryFileName("custom_share_history.x
ml");

    // Set the default share intent
    mShareActionProvider.setShareIntent(getDefaultShareIntent());

    return true;
}
```

[그림 35. setShareHistoryFileName() 을 이용한 커스텀 쉐어 히스토리 만들기]

네비게이션탭(Navigation Tab)

액션바에 네비게이션탭(Navigation Tab)을 이용하면 기존의 탭위젯(Tab widget)보다 훨씬 보기좋은 UI 를 가진다. 시스템이 스크린사이즈에 기반하여 탭을 액션바에 보기좋게 배치시키기 때문이다.

[그림 36. 좁은화면에서 액션바에 포함된 탭]



[그림 37. 넓은 화면에서 액션바에 포함된 탭]

네비게이션탭을 추가하기 위해선 아래 소스코드처럼 액션바에 setNavigationMode 를 Actionbar.NAVIGATION_MODE_TABS 로 설정해 주어야한다.

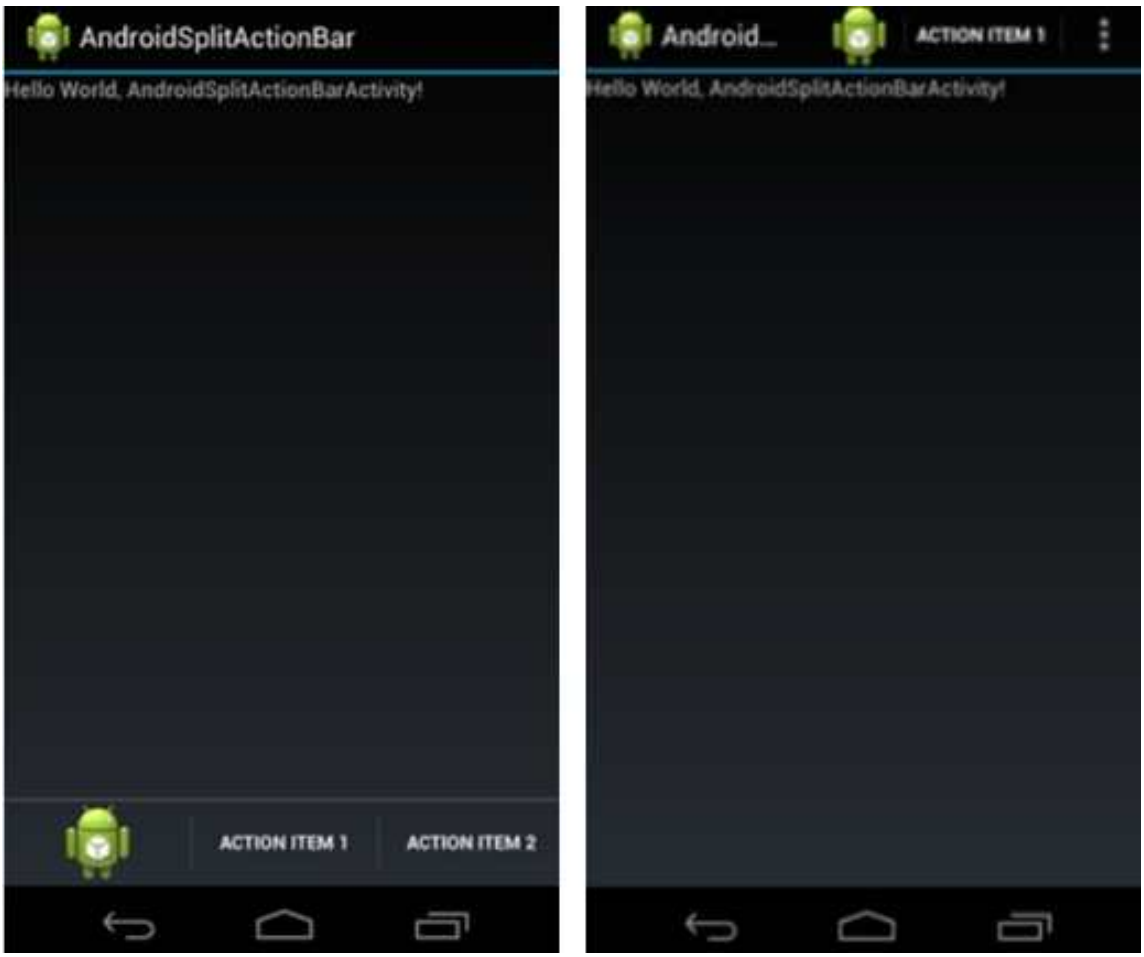
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Notice that setContentView() is not used, because we use the root
    // android.R.id.content as the container for each fragment

    // setup action bar for tabs
    ActionBar actionBar = getActionBar();

actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
    actionBar.setDisplayShowTitleEnabled(false);

    Tab tab = actionBar.newTab()
            .setText(R.string.artist)
            .setTabListener(new TabListener<ArtistFragment>(
                    this, "artist", ArtistFragment.class));
    actionBar.addTab(tab);
```

```
            tab = actionBar.newTab()
                  .setText(R.string.album)
                  .setTabListener(new TabListener<AlbumFragment>(
                              this, "album", AlbumFragment.class));
            actionBar.addTab(tab);
      }
```

[그림 38. 두개의 탭을 포함한 액션바]

분리된 액션바(Split actionbar)

안드로이드 버전 4.0(API level 14) 부터 분리된 액션바 라는 모드가 추가되었다. 이 모드는 아래그림처럼 액션아이템을 하단에 배치시켜서 좀더 효율적으로 좁은 공간을 사용할 수 있도록 제공한다.



[그림 39. 분리된 액션바를 사용한 액티비티(왼쪽), 일반 액션바를 사용한 액티비티(오른쪽)]

분리된 액션바를 사용하려면 아래의 코드처럼 메니페스트의 <activity>엘리먼트에 splitActionBarWhenNarrow 속성을 추가해야한다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.exercise.AndroidSplitActionBar"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:uiOptions="splitActionBarWhenNarrow"
            android:name=".AndroidSplitActionBarActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

[그림 40. AndroidManifext.xml]

[별첨8] 로칼리틱스 (Localytics)

1) 통계 자료의 필요성

모바일 어플리케이션을 기획을 하고 많은 수정을 거쳐가며 출시를 하는데, 출시를 하고 나서 사용자가 우리가 기획한대로 어플리케이션을 쓰고 있는지를 파악을 하는 것이 중요하다. 특정 기능을 공들여서 개발을 하여 완성시켰는데 그것을 사용자가 아무도 사용하지 않는다면 그것은 안 만든 것이나 마찬가지다. 이렇듯 출시 이후에도 사용자가 어플리케이션을 어떻게 사용을 하고 있는지 아는 것이 중요하다. 만약 사용 타겟이 글로벌 하다면, 어느 나라에서 더 많이 쓰고 있는가, 업데이트를 했는데 업데이트 후 어느 정도의 시간이 지난 후 사용자가 업데이트를 시작하였는가 등을 파악하는 것이 중요하다. 그래야만 어느 기능에 보완이 더 필요한지, 어플리케이션에 규모가 큰 업데이트를 했을 때 사용자의 사용 패턴이 어느 정도로 변경이 되었는지 알아보는 것은 아주 중요한 이슈라고 할 수 있다.

추가적으로 이후에 다른 어플리케이션을 개발할 때에도 쌓아놓은 통계 자료들은 성공적인 어플리케이션을 만들기 위한 아주 중요한 자료가 될 것이다.

2) 기존 구글 플레이 통계 자료

구글 플레이 개발자 콘솔에서도 기본적인 사용자의 통계 자료를 제공한다.
활성 기기 설치 수, 총 설치 수, 일일 앱 제거 사용자 수 등 배포하는 쪽에서만 알 수 있는 없어서는 안될 중요한 정보들도 있다. 하지만 이 수치들을 제외한 정보는 실시간이 아닌 하루 간격으로 제공되고 있으며, 다양한 사용자의 어플리케이션 사용 패턴을 파악하기가 힘들기 때문에 운영, 마케팅을 어떤 식으로 해나갈 것인지 결정할 수 있는 다른 정보가 필요하다.

[그림 41. 구글 플레이가 제공하는 통계자료]

3) 모바일 어플리케이션 로그 분석 툴

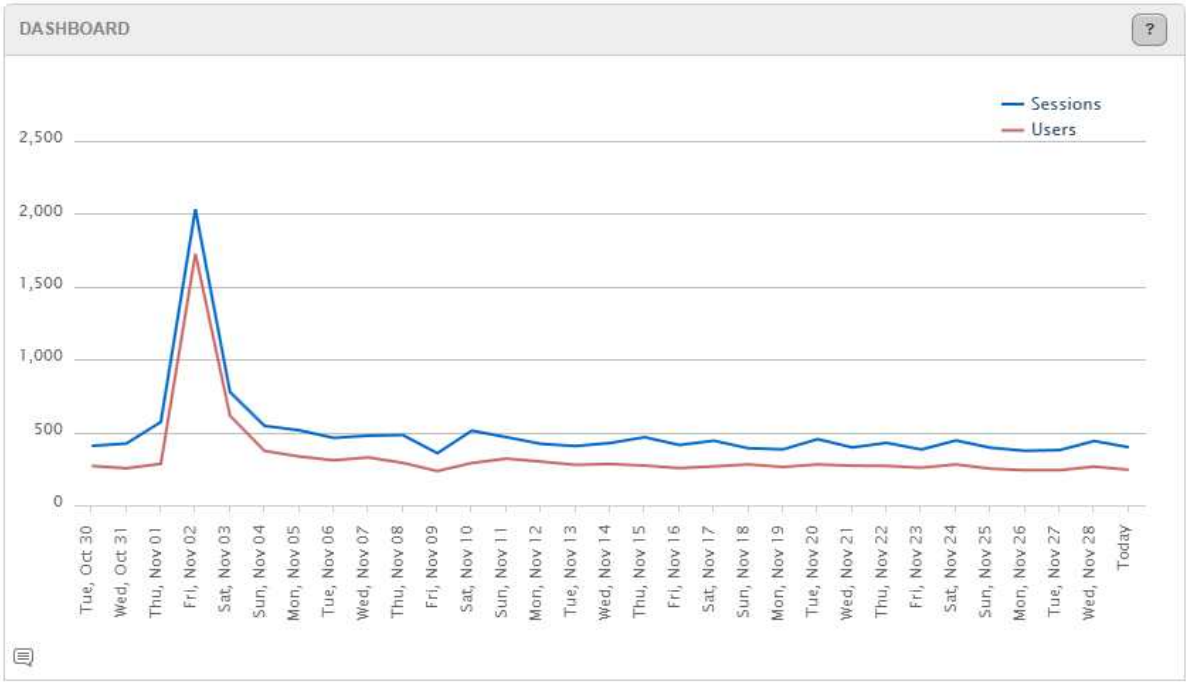
이러한 구글 플레이의 제공자료의 제약을 극복하기 위해 Flurry, Apsalar, Localytics 등등 여러 종류의 어플리케이션 사용자 사용 패턴 분석 툴이 있다.

특히 Flurry 와 Localytics 가 가장 많이 쓰이는데, 제공하는 자료는 거의 비슷하다. 하지만 Flurry 는 자료의 업데이트 간격이 14~15 시간이고, 라이브러리를 jar 파일 형태로 제공을 하고 있다. 반면에 Localytics 는 업데이트 간격이 5 분으로 거의 실시간으로 제공을 하고 있으며, 라이브러리를 소스코드로 개방하여 제공을 하고 있기 때문에 개발자가 상황이나 취향에 맞게 소스코드를 수정할 수가 있다.

4) 제공 자료

Dashboard

메인 화면인 Dashboard 에서는 쌓인 로그 중에서 중요한 내용을 요약해서 보여준다.



[그림 42. Dashborad 그래프]

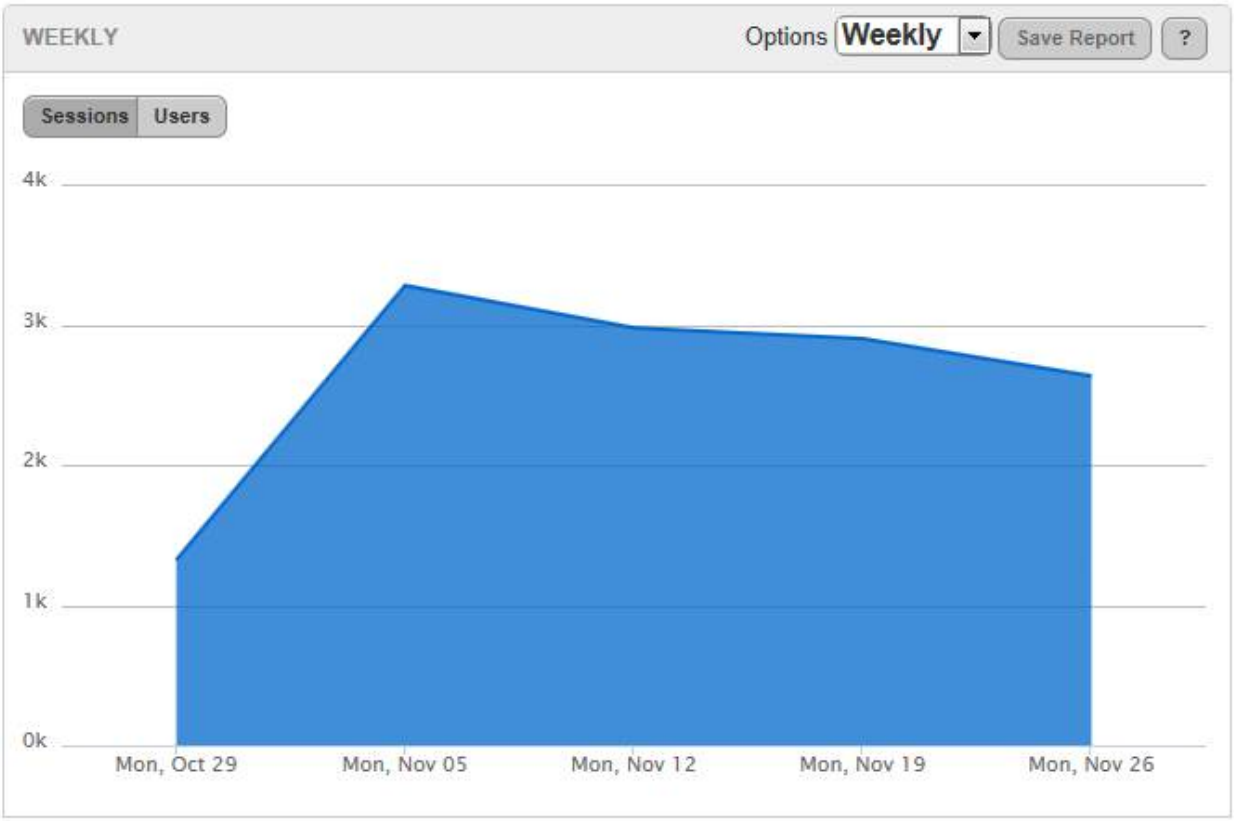
다음은 Dashboard 의 그래프이다. 기본적으로 유저(User), 세션(Session)의 수치가 제공되는데 User 는 어플리케이션을 실행한 기기의 수이고, Session 은 어플리케이션을 실행한 횟수를 나타낸다.

Usage

Usage 에서는 Events 를 제외한 전체적인 어플리케이션의 로그 정보를 확인 할 수가 있다. 모든 리포트는 CSV 파일로 만들어 저장을 해놓을 수도 있으며 위에는 그래프로 전체적인 정보를 확인 할 수 있게 제공하며, 아래에는 자세한 수치를 표로 만들어 제공을 하고 있다.

User & Sessions

시간 별로 User, Sessions 의 유입 분포를 확인 할 수 있는데, 옵션을 활용하여 시간별, 일별, 주별, 월별로 정보를 볼 수 있도록 제공 한다.
하단 그래프에는 현재 조회중인 기간의 User, Sessions 의 총 수치와 옵션에서 선택한 시간별로 자세한 수치를 표로 제공 하고 있다.



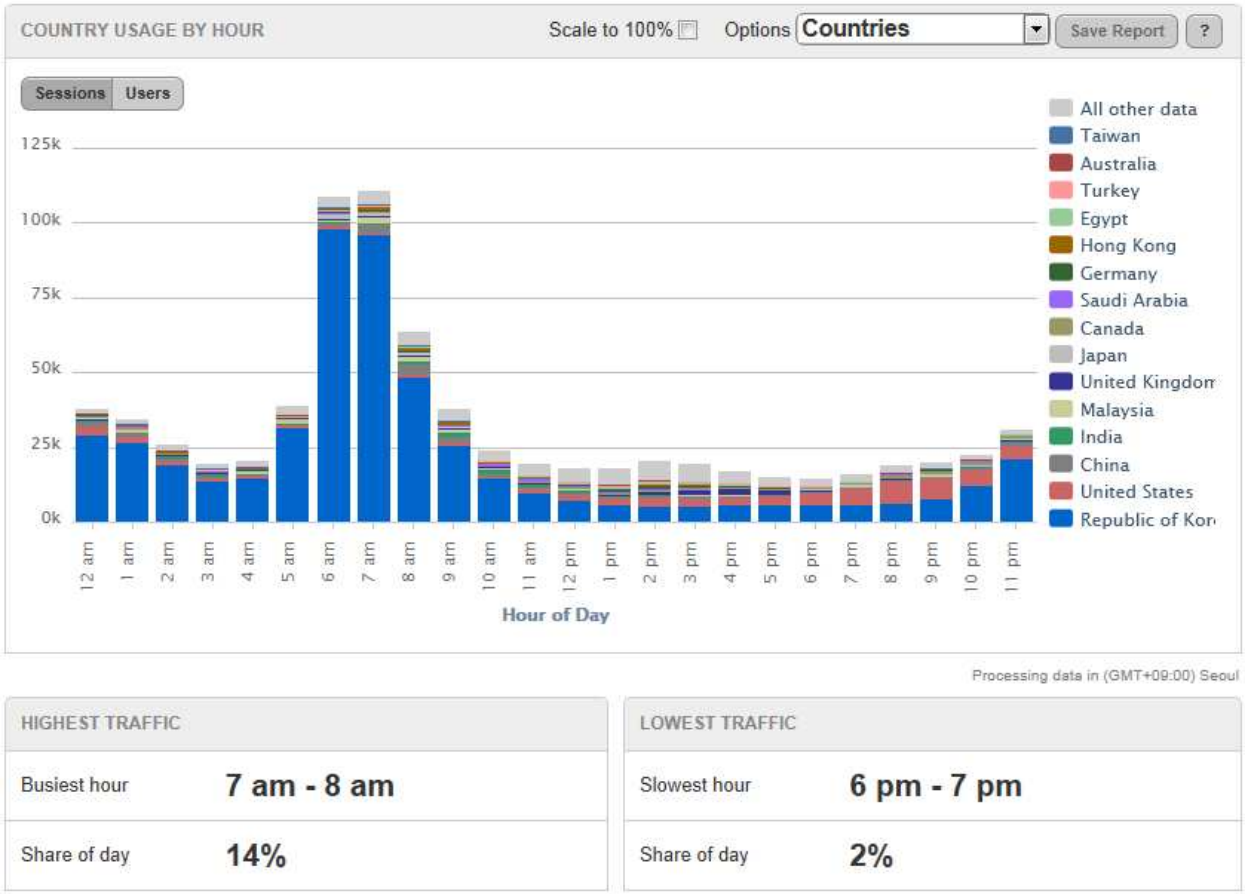
[그림 43.User 와 Session 뷰]

New & Returning

New 는 새롭게 Session 을 연 사용자의 수 이고, Returning 은 예전에 Session 을 연적이 있는 사용자의 수 이다. 옵션에서는 시간별, 일별, 주별, 월별로 정보를 볼 수 있도록 제공한다.
아래 표에서는 조회중인 기간에서의 User 수와 Session 의 총합을 보여주고, 새로운 사용자의 비율과 새로운 사용자에 의해 열린 Session 의 비율을 보여준다.

Daypart

매 시간 별로 User 와 Session 의 수치를 보여준다. 어느 시간대에 어플리케이션이 많이 사용되는지 알 수 있도록 도와준다. 옵션에서는 시간별, New & returning, 국가별, 어플리케이션버전 별로 정보를 확인할 수 있다.
아래 표에서는 가장 많이 어플리케이션을 사용한 시간대, 가장 적게 어플리케이션을 사용한 시간대를 보여주며, 각각의 비율도 확인 가능하다.

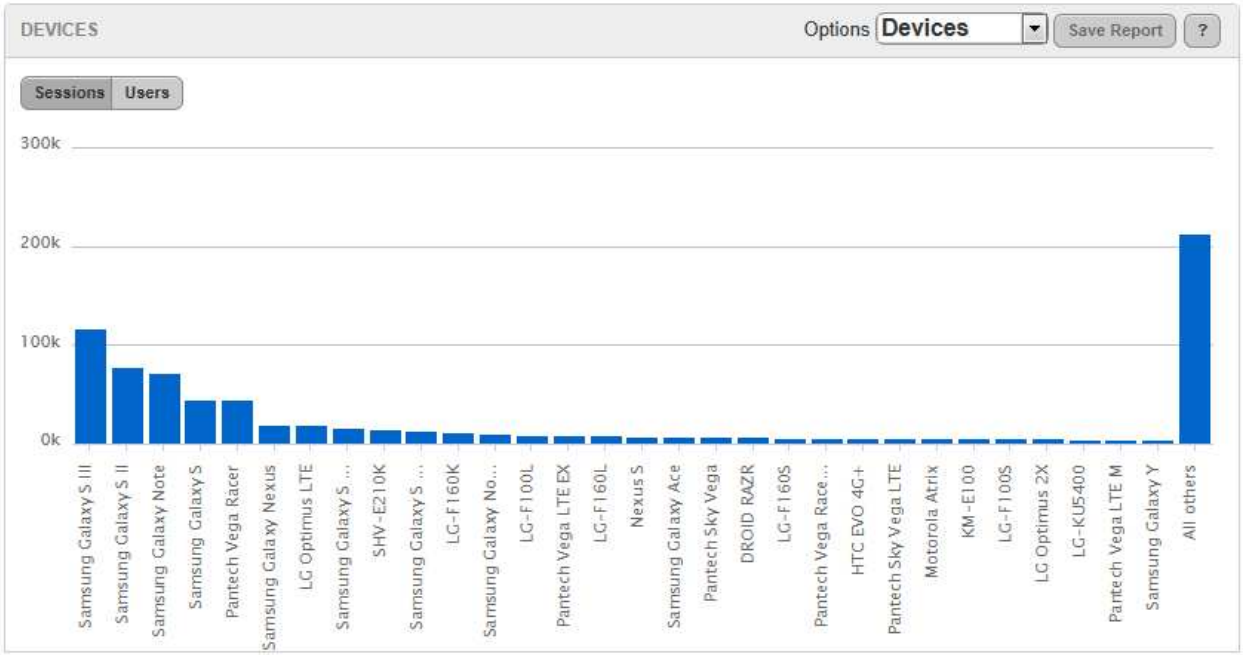


[그림 44.시간대별 사용 나라 비율 리포트]

Device

Session 을 연 기기들의 수치를 확인 할 수 있다. 옵션을 통신사별, 국가별, 어플리케이션 버전별, 시간별로 그래프를 볼 수 가 있으며, 어플리케이션을 실행한 모델의 총수도 확인 가능하다.

이 그래프를 활용하여 어느 나라에서 어느 기기가 더 많이 쓰이는지 파악 할 수 있으며, 어플리케이션을 서비스 할 때 모든 기기에서 호환을 할 시간적 여유가 안되거나, 상황이 안될 때에는 비율을 파악하여 어느 기기에 초점을 맞춰서 개발을 할 지 파악할 수 있다.



[그림 45.디바이스 분포도]

Carrier

통신사 별 분포 그래프를 볼 수가 있으며, 옵션에서 국가별, 기기별로 그래프를 볼 수 있다.

Country

국가 별 분포 그래프를 볼 수가 있으며, 옵션에서 통신사별, 기기별로 그래프를 볼 수 있다.

OS Version

안드로이드 OS 버전 별 분포 그래프를 볼 수가 있다. 옵션에서는 기기별, 어플리케이션 버전별로 상세한 그래프를 볼 수 있다.

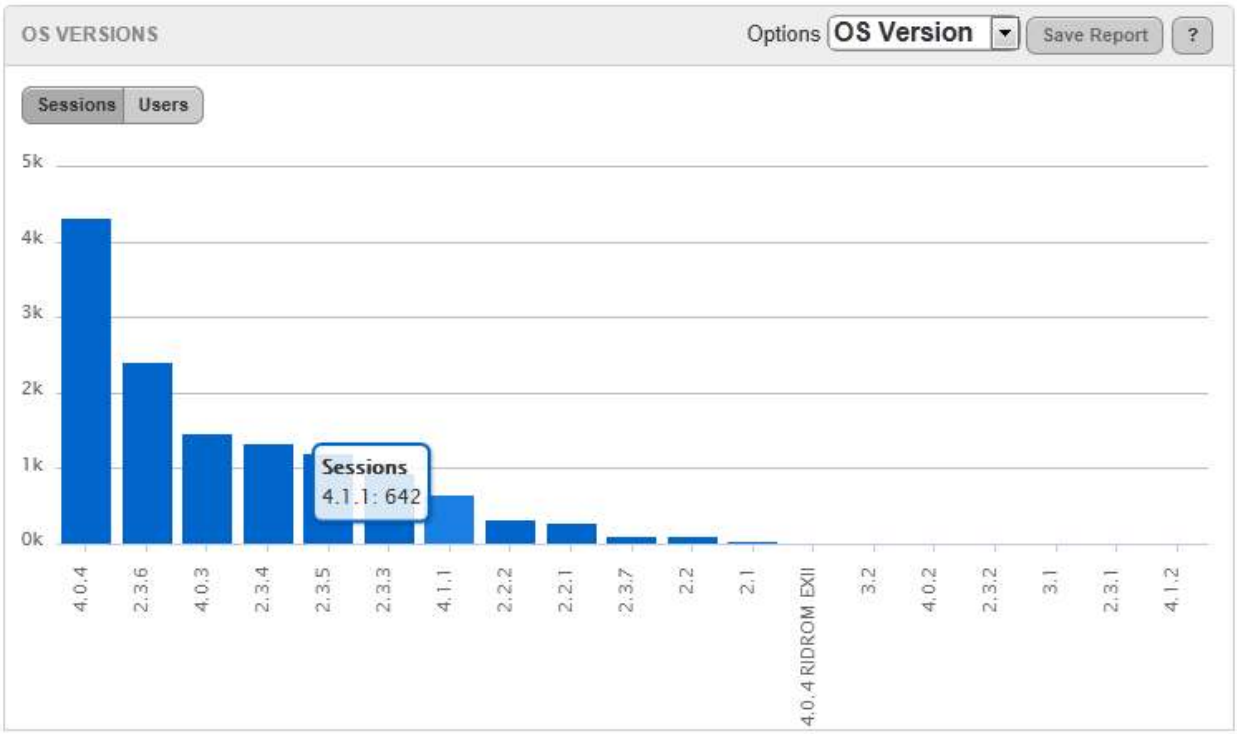


CHART DATA

Export to CSV

OS Version	Sessions	% of total
4.0.4	4,319	33%
2.3.6	2,399	18%
4.0.3	1,457	11%
2.3.4	1,333	10%

[그림 46.안드로이드 OS 버전별 사용 현황]

App Version

어플리케이션의 버전 분포 그래프이다. 옵션을 통하여 일별, 주별, 월별로 그래프를 볼 수 있다.

이 그래프를 활용하여 어플리케이션 업데이트 시 사용자가 어느 정도로 업데이트를 꾸준히 해주는지 파악이 가능하다.

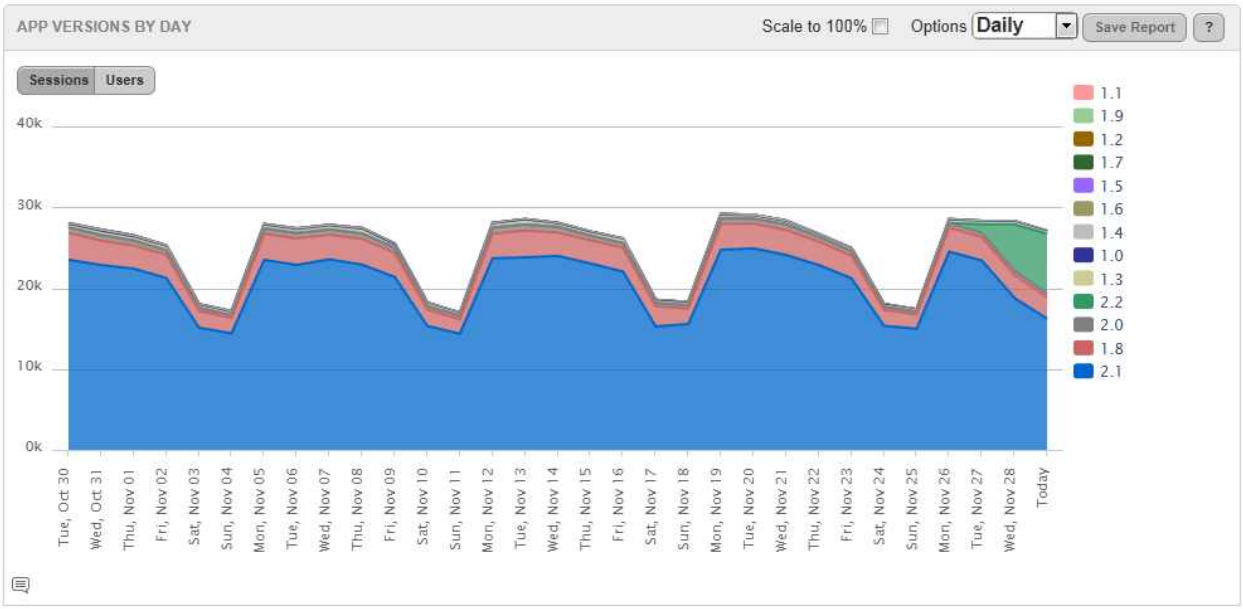


CHART DATA — Export to CSV

Days	2.1	1.8	2.0	2.2	1.3	1.0	1.4	1.6	1.5	1.7	1.2	1.9	1.1	Total
Tue, Oct 30	23,520	3,342	652		192	116	110	45	55	14	1	4		28,051
Wed, Oct 31	22,862	3,028	685		241	154	140	63	56	13	3	2		27,247
Thu, Nov 01	22,410	2,869	604		241	133	118	69	73	11	10	11		26,549
Fri, Nov 02	21,259	2,893	538		244	104	114	56	43	10	9			25,270
Sat, Nov 03	15,118	2,079	391		160	100	70	58	36	8	2			18,022
Sun, Nov 04	14,417	1,893	336		174	117	121	47	40	12	5		2	17,164

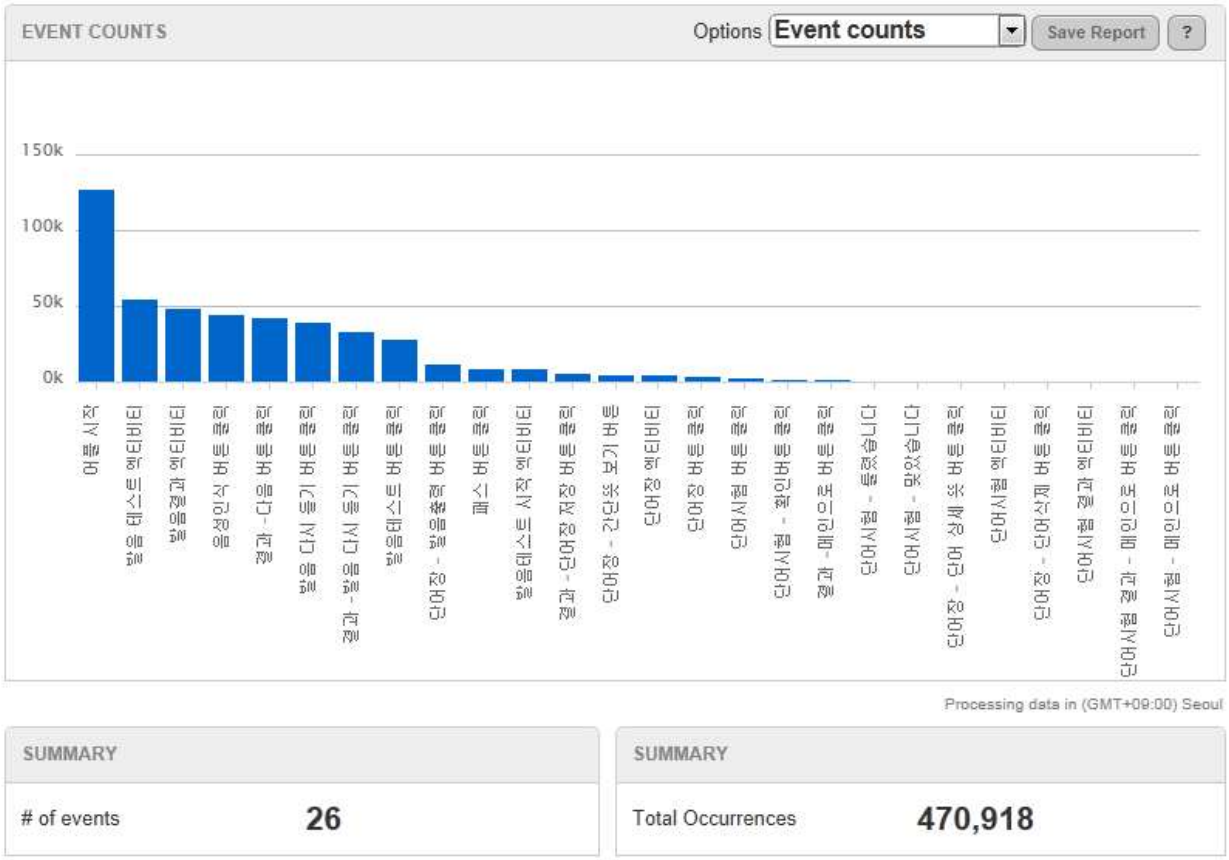
[그림 48.앱 버전별 사용 현황]

Events

Events 는 개발자가 소스코드 내부에 태그를 달아 개발자가 원하는 위치에 원하는 정보를 가져올 수 있다. 예를 들면, 어떤 엑티비티로 넘어 왔을 때, onCreate 함수 내부에 태그를 달아서 해당 액티비티에 들어오는 비율이 어느 정도인지 파악이 가능하고, 버튼이벤트 리스너 내부에 태그를 달아 해당 기능이 어느 정도 사용이 되고 있는가 파악이 가능하게 된다.

Event Overview

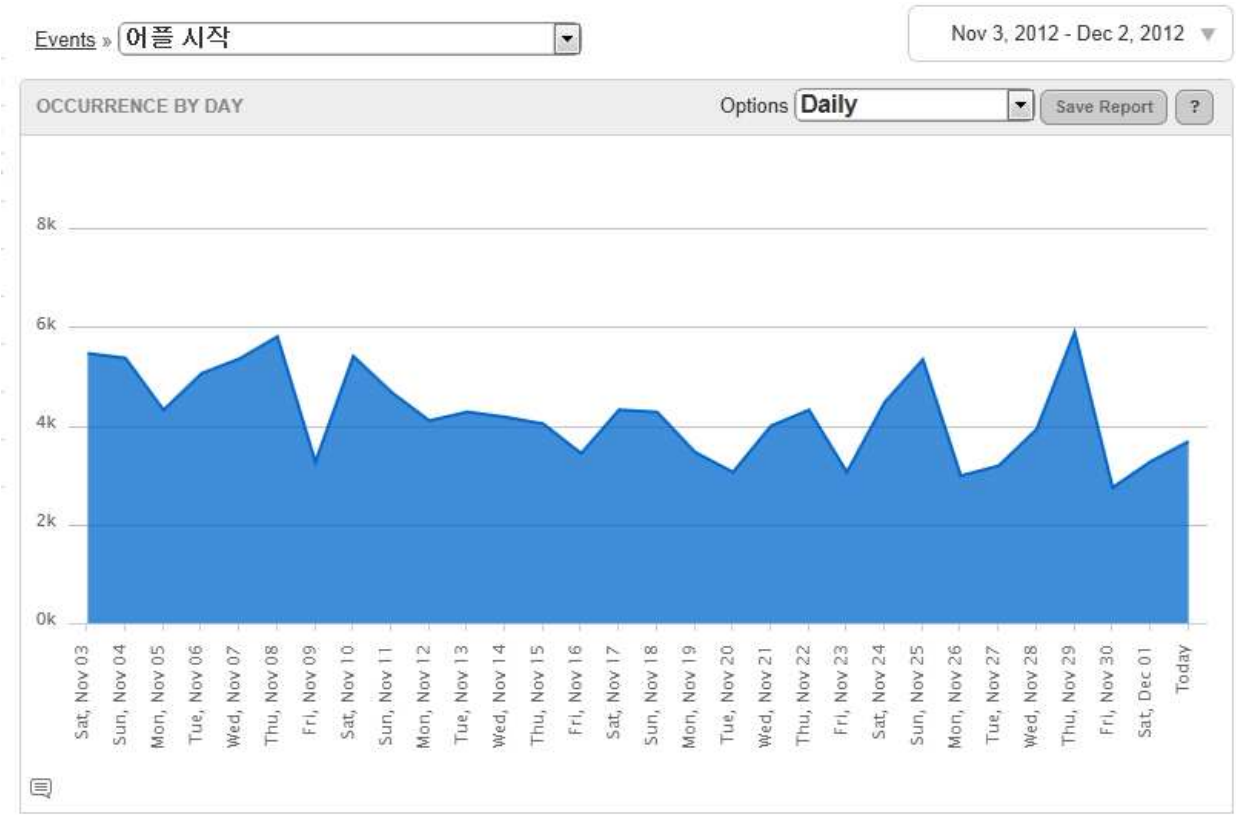
어플리케이션에 있는 모든 태그의 분포 그래프이다. 옵션을 통해서 유저별, 세션별, New & Session 별로 그래프를 볼 수 있다.



[그림 49.이벤트 태그뷰]

Event Detail

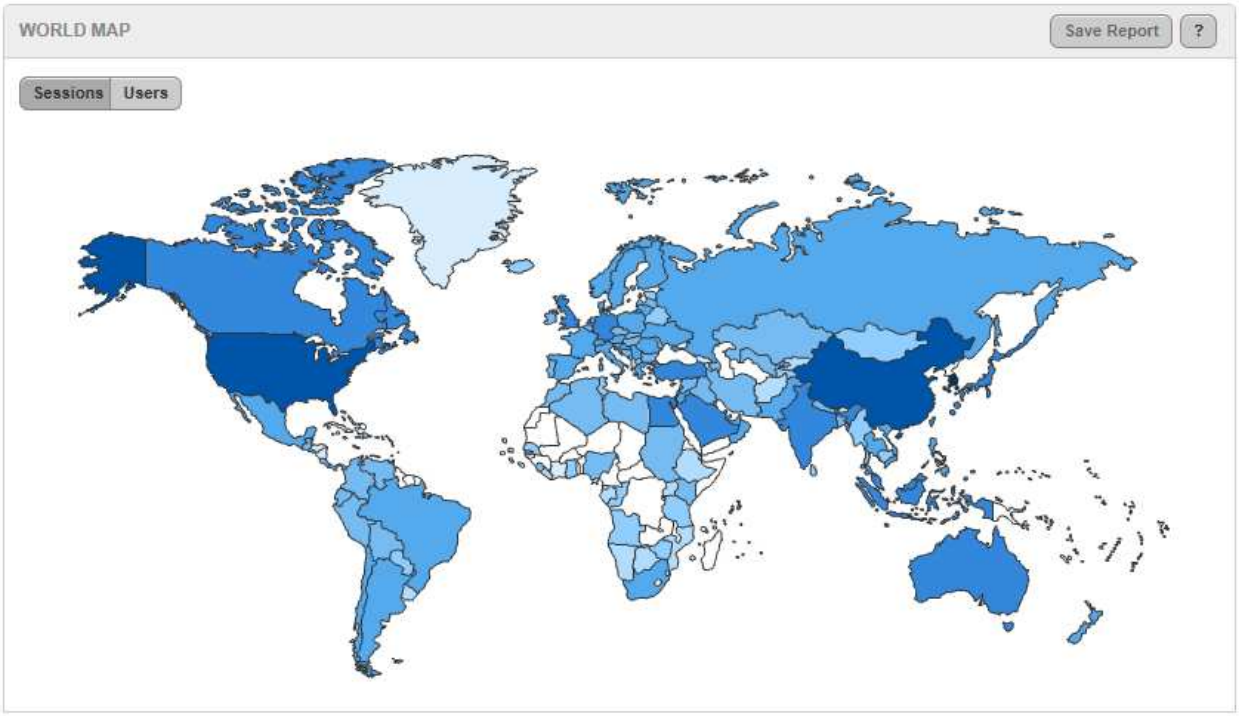
각 태그 별로 상세한 수치를 볼 수 있다. 옵션에서도 시간대별, 기기별등 많은 옵션들로 각 분포도를 확인 할 수 있다.



[그림 50.이벤트별 시간대 사용 현황]

Maps

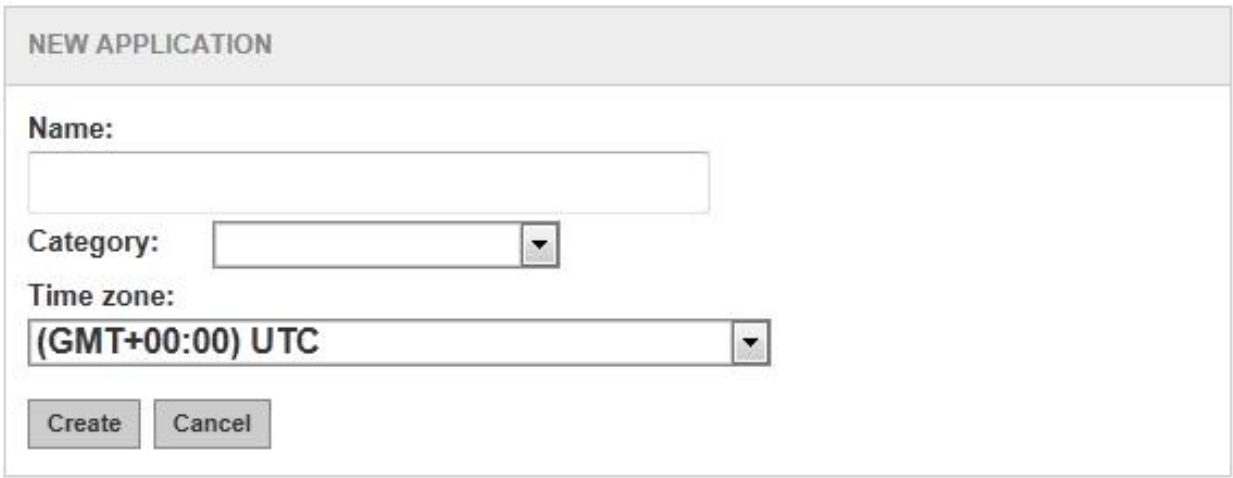
세계지도를 통하여 Session 이 실행된 지역을 보여준다. 나라의 색이 진할수록 어플리케이션을 사용하는 사용자가 많다.



[그림 51.세계 사용 현황]

5) 설치법

우선 가장 먼저 Localytics 홈페이지 (http://www.localytics.com/)에서 회원가입을 한 후 어플리케이션을 등록하여 키를 발급 받아야한다.
Administration 탭에서 NewApplication 버튼을 눌러 어플리케이션을 등록한다.



[그림 52.설치법]

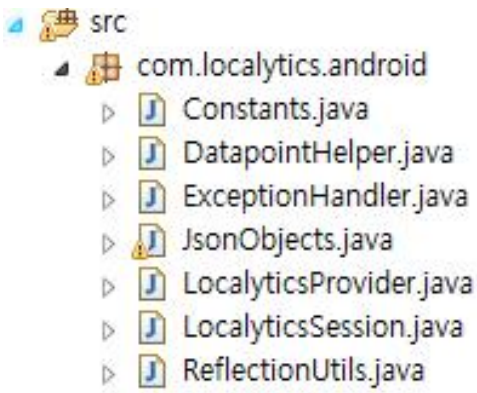
- Name : Localytics 에서 사용할 해당 어플리케이션의 이름

- Category : 해당 어플리케이션의 카테고리
- Time zone : 해당 어플리케이션 로그를 보여줄 때 사용할 시차 설정

등록을 하게 되면 아래 그림과 같이 등록한 어플리케이션의 정보가 뜬다. 발급 받은 키와 홈페이지에서 제공하는 안드로이드 라이브러리를 어플리케이션 소스에 추가하여 적용시키면 된다.

[그림 53. 발급 받은 키]

Localytics 홈페이지에서 안드로이드 라이브러리를 다운 받아 압축을 푼 소스코드를 어플리케이션 src/폴더에 추가 한다.



[그림 54. Localytics 에서 제공하는 소스]

소스코드를 복사했으면 이제 로그를 기록할 엑티비티에 클래스를 추가한다.

```
import com.localytics.android.*;
```

클래스 내부에 세션 객체를 생성한다,

```
public static LocalyticsSession localyticsSession;
```

그리고 onCreate 함수내에서 세션 객체로 세션을 열고, 데이터를 업로드 하면 된다.

```
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  this.localyticsSession = new LocalyticsSession(
  this.getApplicationContext(), "APP KEY");

  this.localyticsSession.open(); // open the session
  this.localyticsSession.upload(); // upload any data
}
```

그리고 추가적으로 기기가 어플리케이션을 완전히 종료된 게 아니라 홈 버튼 등을 통하여 잠시 어플리케이션을 빠져나올 경우를 대비해 onPause 메소드에서 세션을 닫고, 현재까지 받은 정보를 업로드 시켜야 한다. 마찬가지로, 어플리케이션이 다시 실행 되는 경우를 대비하여 onResume 메소드에서 세션을 다시 열도록 구현해 놓아야 한다.

```
@Override
public void onPause() {
  localyticsSession.close();
  localyticsSession.upload();
  super.onPause();
}

@Override
public void onResume() {
  super.onResume();
  localyticsSession.open();
}
```

이제 Localytic 를 사용할 준비는 다 되었다.
위의 설치법대로만 해도 제공하는 정보의 대부분을 얻을 수 있다. 하지만, 부가적으로 tagEvent 를 통하여 개발자가 원하는 태그를 원하는 위치에 달수도 있다.

```
localyticsSession.tagEvent("TagEvent 남김");
```

소스코드에서 tagEvent 메소드가 실행이 되면 Localytics 에 카운팅이 되고, 이 정보를 활용하여 개발자가 원하는 다양한 사용자의 패턴이 파악 가능하다.

아래는 해당 엑티비티의 클래스 전체 구조이다.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.view.View;
import android.view.View.OnClickListener;
import com.Localytics.android.*;
public class SkeletonView extends Activity
{
 private final static String LOCALYTICS_APP_KEY
 ="KEY_GENERATED_ONLINE";
 private final static String EVENT_TEST_BUTTON = "Test Button";
 private LocalyticsSession localyticsSession;
 private Button testButton;
 public void onCreate(Bundle savedInstanceState)
 {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        this.localyticsSession = new LocalyticsSession(
        this.getApplicationContext(),
        SkeletonView.LOCALYTICS_APP_KEY);
        this.localyticsSession.open();
        this.localyticsSession.upload();
        // This demo assumes a button called TestButton has been
```

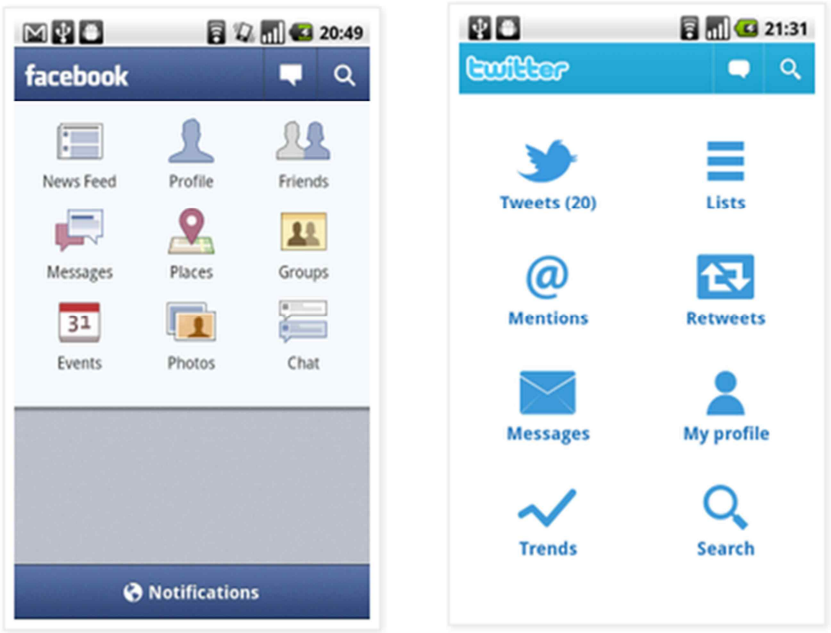
```
defined
        // in the main XML layout.
        this.testButton = (Button)findViewById(R.id.TestButton);
        this.testButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v)
            {
 localyticsSession.tagEvent(SkeletonView.EVENT_TEST_BUTTON);
            }
        });
 }
 public void onResume()
 {
        super.onResume();
        this.localyticsSession.open();
 }
 public void onPause()
 {
        super.onPause();
        this.localyticsSession.close();
        this.localyticsSession.upload();
 }
}
```

6) 문제점 및 이슈

위와 같이 간단하게 사용자의 기기정보와 사용패턴 등을 파악 할 수 있지만, 서버 쪽으로 로그 데이터를 보내야 하기 때문에 반드시 INTERNET 퍼미션이 필요하므로, 꼭 네트워크연결이 필요 없는 어플리케이션이라 하더라도 반드시 사용자에게 데이터이용료가 부과된다는 사실을 공지 하여야 한다.
그리고 기기일렬번호, 휴대폰번호 등 그리고 각 사용자를 식별할 수 있는 개인정보는 태그를 활용하여 보내면 법적으로 큰 문제가 발생할 수 있으므로 유의하여 사용해야 한다.

[별첨9] Side Navigation

1) 기존 유아이의 문제점



[그림 54. Side navigation 을 적용하기전의 UI]

대시보드는 한 화면에 다양한 정보를 관리하고 찾을 수 있도록 하는 사용자 인터페이스 중 하나이다. 메시지 전송, 사진 업로드, 미디어 파일 관리 등의 다양한 컨텐츠를 한 화면에서 관리하고 있기 때문에 사전 기능을 쉽게 탐색할 수 있을 뿐만 아니라 주요 기능에 쉽고 빠르게 접근할 수 있다.
이러한 장점으로 기존의 어플리케이션의 일반적인 유아이로 활용되었고, 소셜 네트워크의 대표적인 트위터나 페이스북에서도 활용되었다.

2) 대시 보드의 한계점

[그림 55. 대시보드 UI]

하지만 대시보드 유아이는 사용자가 응용 프로그램의 기능을 이용하기 위해서는 여러 과정을 거쳐야 하고, 다른 기능을 이용할 때는 어플리케이션의 첫 화면으로 돌아 가야하는 번거로움이 있다.

좀 더 자세하게 위의 와이어프레임 그림을 통해 살펴보면, 사용자가 어플리케이션의 한 기능을 사용하기 위해서는 대시보드 유아이에서 메뉴를 선택하고 세부 기능을 입력한 후에야 해당 기능을 사용할 수 있다. 또한 다른 기능을 이용하기 위해서는 홈 버튼이나 백 버튼을 눌러 첫 화면으로 돌아간 후 위의 과정을 똑같이 반복해야 한다.

즉 대시 보드 형태의 어플리케이션 구조는 사용자들에게 불편함과 번거로움을 가져다 주었다.

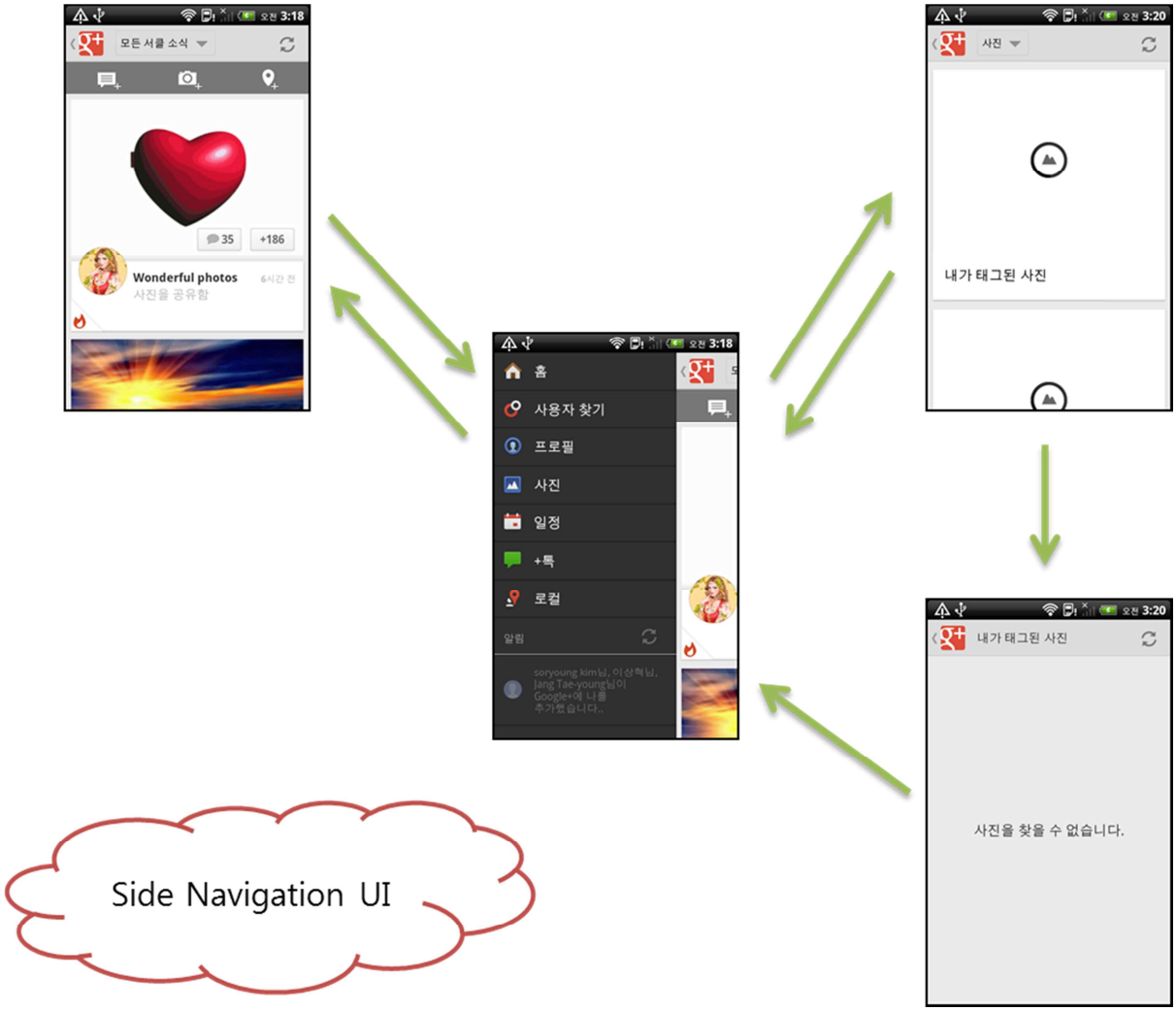
3) 사이드 네비게이션이란

사이드 네비게이션은 화면의 옆면을 활용한 유아이 패턴이다. 모바일이라는 사이즈 한계를 극복하여 보이지 않는 사이드 측면을 활용했기에 때문에 화면 구성을 보다 효과적으로 사용할 수 있다.

사이드 네비게이션의 대표적인 예는 현재 버전의 페이스북 어플리케이션이다. 기존에는 대시 보드 형태로 메뉴를 구성했지만, 버전이 업데이트되면서 페이스북은 사이드 네비게이션 형태의 유아이를 제공하고 있다. 측면을 통해 서브 메뉴를 제공할 뿐만 아니라 사용자는 화면을 종료하지 않고도 애플리케이션의 메뉴를 사용할 수 있다.

[그림 56. Side Navigation UI]

4) 사이드 네비게이션의 흐름도

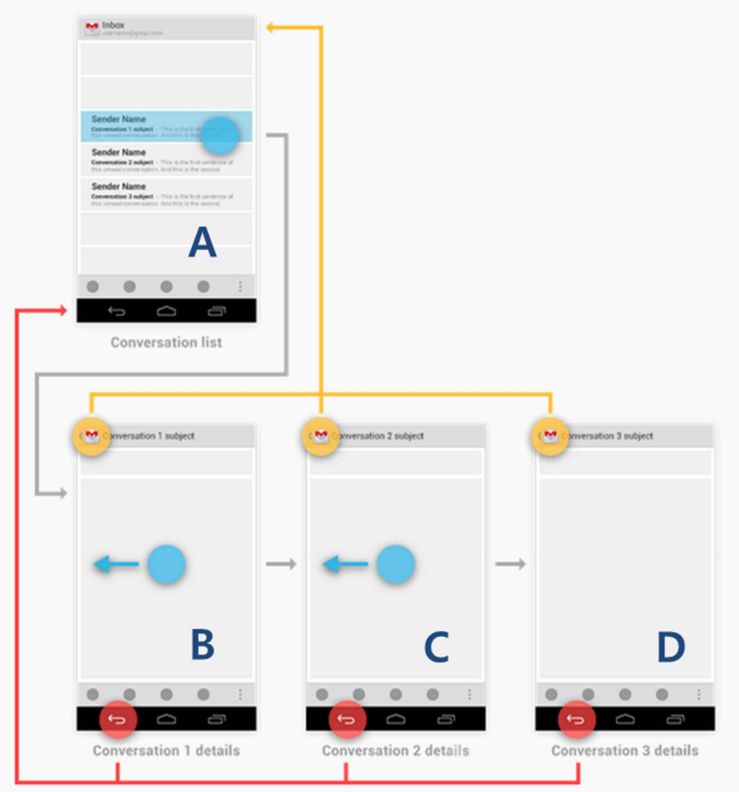


[그림 57. Side Navigation UI]

사이드 네비게이션은 기존의 대시보드의 한계점을 해결해주는 유아이라고 할 수 있다. 어플리케이션의 화면을 이동하지 않고서도 사용자는 주요 기능들을 바로 확인할 수 있기 때문이다. 그러므로 사용자는 응용 프로그램을 시작하게 되면 대시보드의 메인 화면을 보는 대신에 주요 기능을 실행시킬 수 있고, 화면의 전환없이 메뉴를 선택할 수 있게 되었다.
위의 와이어프레임을 통해 사이드 네비게이션의 흐름도를 살펴보면, 먼저 사용자는 어플리케이션을 실행하였을 때 메뉴 구성 화면 대신 하나의 주요 기능를 사용할 수 있다. 또한 다른 기능을 사용하고자 하면 사이드 네비게이션을 통해 어플리케이션의 기능을 살펴보고 그 중 하나의 기능을 선택하여 이용할 수 있다.

5) 사이드 네비게이션의 예시

A. 사이드 네비게이션의 문제점



[그림 58. Side Navigation UI 의 문제점]

안드로이드 개발자 사이트에서는, 사용자들이 버튼에 대해 예측하고 사용할 수 있도록 Back 버튼과 Up 버튼 등의 디자인 패턴에 대해 정의하였다.
먼저 Back 버튼은 일반적인 뒤로 가기 버튼으로, 사용자가 최근에 작업한 화면의 전 단계를 보여주며 스크린과 스크린 사이의 시간적 관계를 기준으로 동작하게 된다.
그리고 스크린의 왼쪽 상단에 있는 Up 버튼은 스크린 사이의 계층 관계에 따라 어플리케이션을 탐색하고 이동하기 위한 버튼이다.

위의 그림을 통해 예를 들어보자. A->B->C->D 순으로 화면이 구성되었을 때, D 화면에서 Back 버튼을 누르게 되면 C 화면으로 이동하고, C 화면에서 Back 버튼을 누르게 되면 B 화면으로 이동하게 된다. 즉 Back 버튼은 시간순으로 버튼이 동작한다. 또한 Up 버튼은 B,C,D 화면에 상관없이 B,C,D 화면의 부모 화면인 A 화면으로 돌아간다. 즉, 안드로이드 유아이에서 Up 버튼은 스크린 사이의 계층 관계에 따라 동작하게 되는 버튼으로 정의하였다.

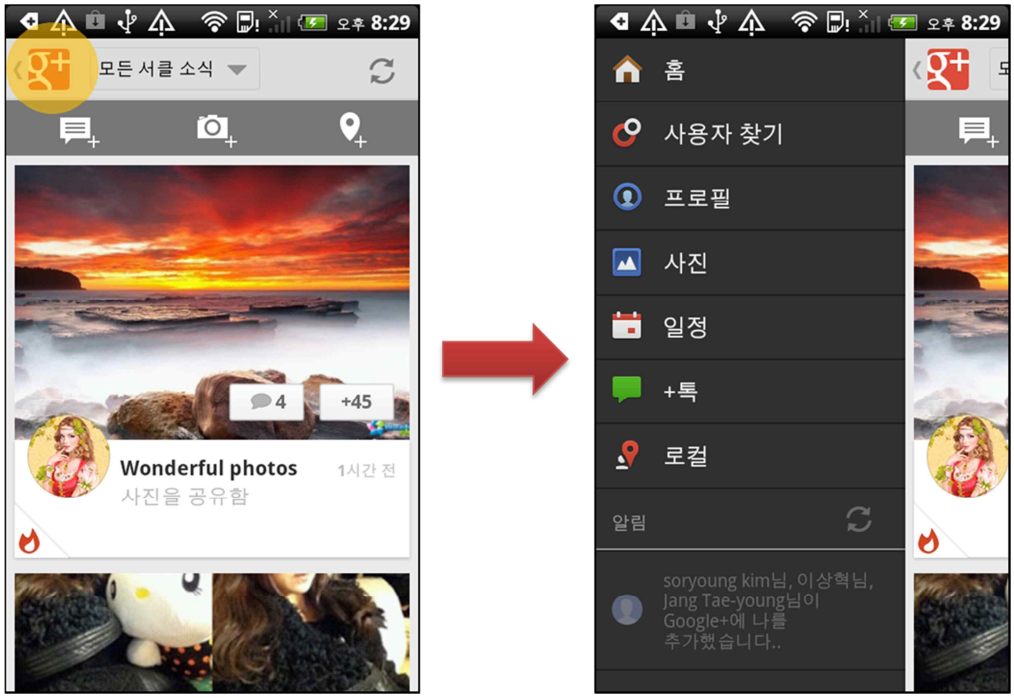
하지만 Facebook 이나 google+ 등의 어플리케이션에서는 안드로이드에서 정의한 디자인의 패턴과 다르게 동작함으로써 사용자들에게 혼돈과 불편함을 주고 있다.

1) Facebook Application

[그림 59. 페이스북의 Side Navigation]

소셜 네트워킹 서비스를 위한 페이스북 어플리케이션에서도 사이드 네비게이션 유아이를 사용한다. 사용자는 왼쪽 상단의 버튼을 통해 페이스북의 세부 메뉴를 선택할 수 있다.

2) Google+ Application



[그림 60. Google + 의 Side Navigation]

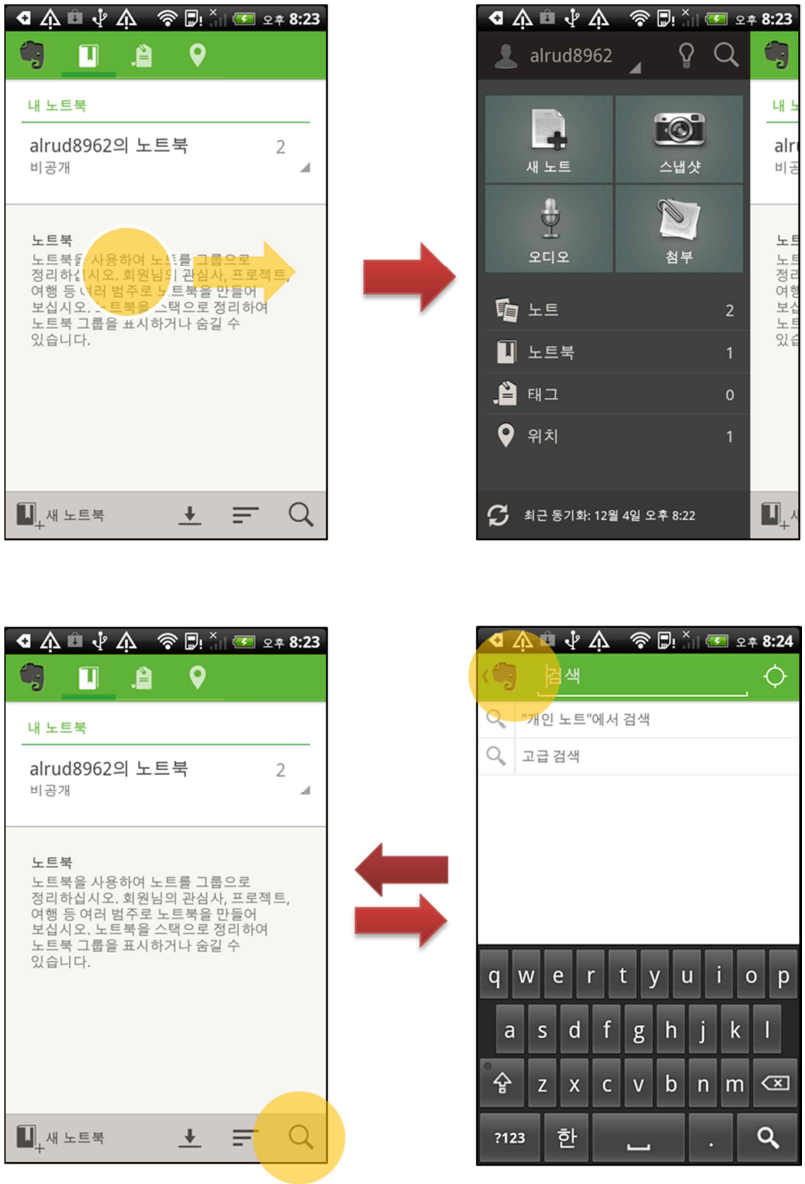
Google+ Application 또한 사이드 네비게이션을 통해 메뉴를 구성하고 있으며, 페이스북과 마찬가지로 왼쪽 상단의 버튼을 통하여 google+의 세부 메뉴를 확인할 수 있다.

위의 예제를 통해 볼 수 있듯이 Facebook 과 Google+ 어플리케이션의 UX 는 안드로이드 플랫폼에서 정의한 Up 버튼의 패턴과 다르게 동작할 뿐만 아니라 google+에서는 Up 버튼의 아이콘 옆에 왼쪽 방향의 캐럿이 포함되어 혼돈을 가중시켰다.
위에서 보여준 두 어플리케이션은 안드로이드 플랫폼에 익숙한 사용자들의 경험을 고려하지 않은 채 설계된 디자인이며, 이러한 디자인은 일부 사용자들에게는 불편함과 혼돈을 주고 있다.

B. 사이드 네비게이션의 개선

잘못 설계된 UX 는 사용자들에게 혼돈을 가져다 줄 수 있지만 위에서 설명한 UX 를 개선한다면, 사이드 네비게이션은 사용자들에게 편리함을 주는 UI 로 활용될 수 있다.
아래의 어플리케이션 예제를 통하여, 위의 문제점을 개선하는 방법에 대해 알아보자.

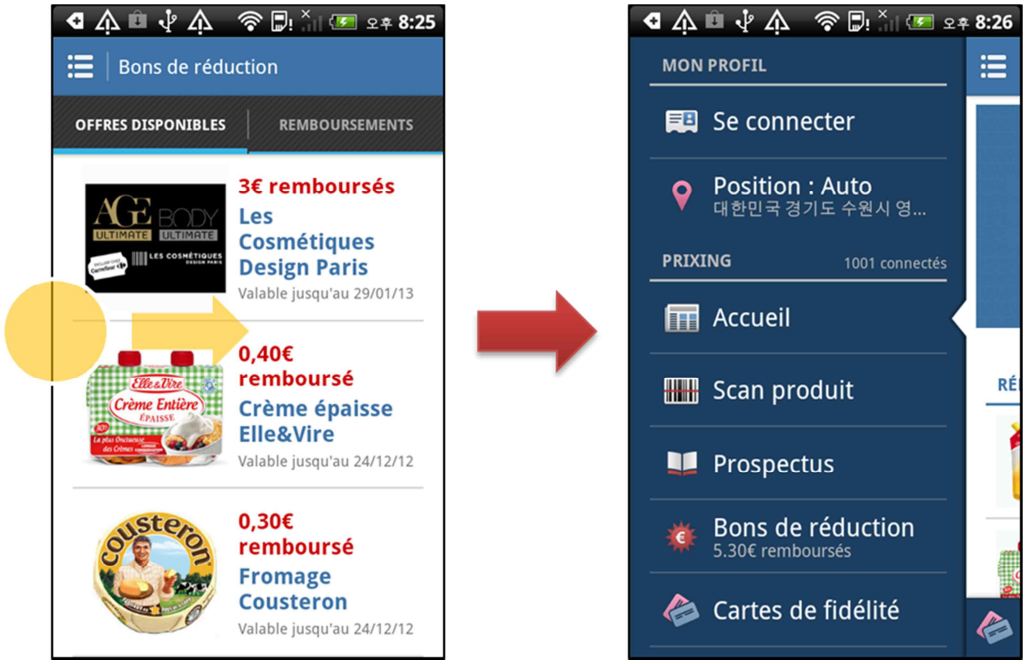
1) Evernote



[그림 61. 에버노트의 Side Navigation]

Evernote 어플리케이션은 화면을 오른쪽으로 스크롤하는 방식으로 사이드 네비게이션을 활용하였으며, 사이드 화면을 통한 메뉴 구성 이외에도 동기화 기능 및 버튼 효과를 포함시켰다. 또한 진동 효과를 주어 사이드 메뉴가 열리고 닫힘을 표현한다.

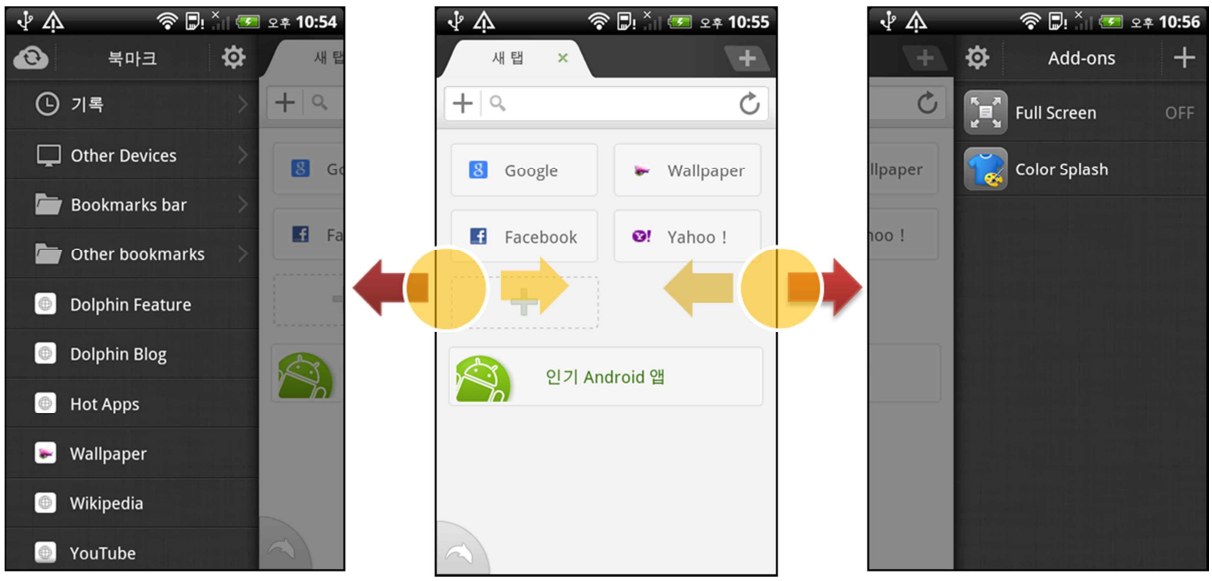
2) Prixing



[그림 62. Prixing 의 Side Navigation]

Prixing 어플리케이션은 화면의 왼쪽 부분을 오른쪽으로 스크롤하는 방식으로 사이드 네비게이션을 활용하였다. 또한 화살표를 통해 현재 사용되고 있는 화면의 메뉴를 알려주고 있다.

3) Dolphin Browse HD



[그림 63. 돌핀 브라우저의 Side Navigation]

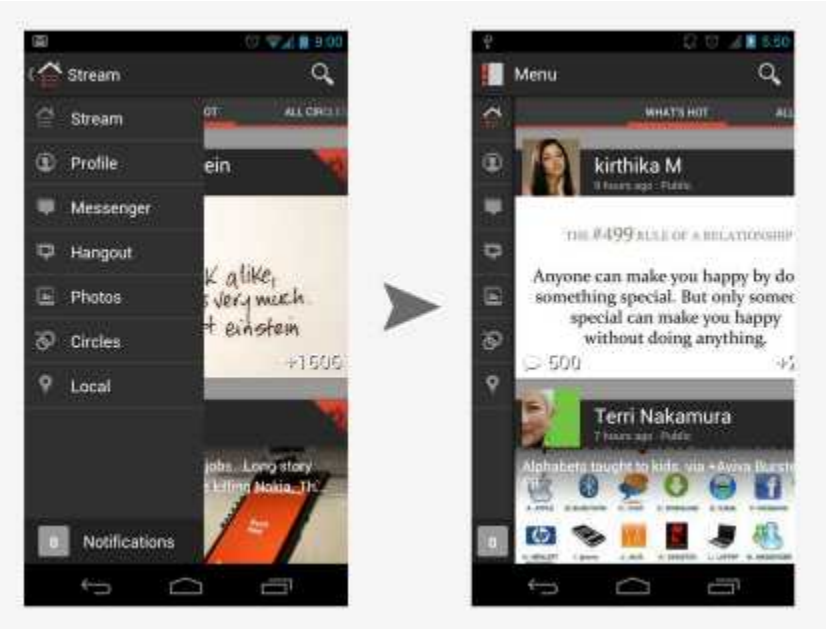
돌핀브라우저 HD 어플리케이션은 앙방향을 통하여 사이드 네비게이션을 사용할 수 있으며, 모바일이라는 제약된 환경에서 스크린 공간을 최대로 확대시켰다. 또한 양방향 사이드 네비게이션 기능을 통해 북마크와 애드온 등의 주요 기능을 빠르게 접할 수 있다는 장점도 있다.

위의 예제 어플리케이션은 안드로이드 플랫폼에서 제공하는 Up 버튼 대신에 화면의 일부를 스크롤하는 방식으로 Facebook 이나 google+가 가지는 사이드 네비게이션 UX 의 문제점을 해결할 수 있었다.

6) 사이드 네비게이션의 개선 방향

사이드 네비게이션은 Facebook 과 같이 다양한 화면을 제공하거나, Evernote 처럼 다양한 유형의 기능을 제공하고 있을 때 유용하게 사용할 수 있는 UI 이다.
하지만 사이드 네비게이션은 안드로이드 플랫폼의 UI 패턴에 정의되어 있지 않아, 사이드 네비게이션을 처음 접하는 사용자들에게는 다소 사용하기 어려운 UI 로 다가갈 수 있다.
사용자가 직관적으로 인식할 수 있는 화면이 될 수 있도록 사이드 네비게이션 UI 패턴의 개선 방향에 대해 알아보자.

개선 방향 1) Side Navigation Panel 을 통한 개선



[그림 64. Side Navigation Panel]

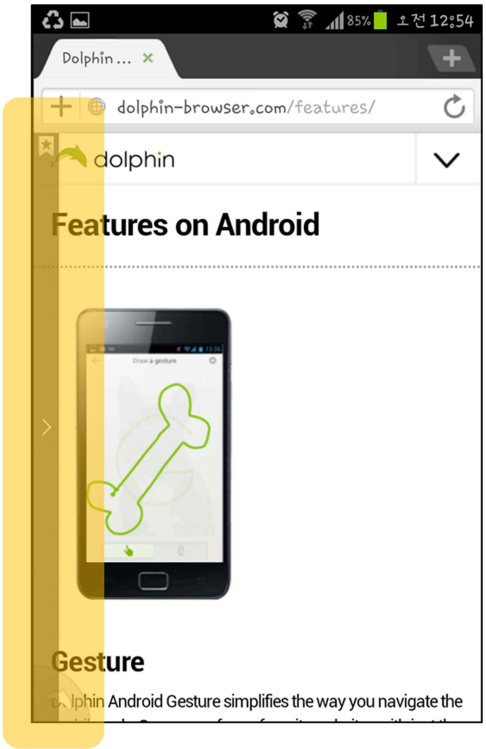
아이콘과 텍스트를 통해 메뉴를 구성하는 대신에, 아이콘만으로 메뉴를 표현한 Side Navigation Panel 을 사용한다. 이 방법은 화면의 최소한의 공간만 차지할 뿐만 아니라 사용자가 메인 화면에서 세부 기능을 바로 접근할 수 있다는 장점이 있다.

개선 방향 2) Side Navigation 을 위한 힌트 제공

사이드 네비게이션은 안드로이드 플랫폼의 UI 패턴에 정의되어 있지 않으며 UI 를 알려주는 명확한 아이콘 또한 존재하지 않는다. 때문에 이 UI 를 처음 접하는 사용자들이 사이드 네비게이션의 기능을 인식하는 데는 어려움이 따른다

[그림 65. Side Navigation 을 위한 힌트를 제공하는 뷰 1]

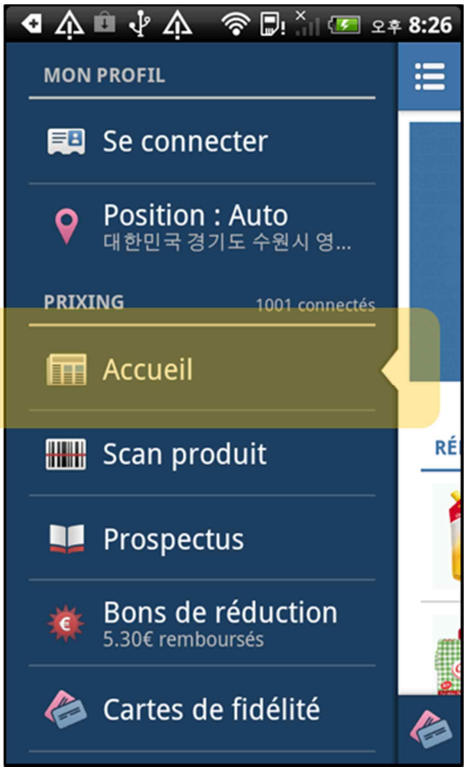
사이드 네비게이션을 인식하기 위해서는, 처음 어플리케이션을 실행하였을 때 사용자 가이드를 제공하여 인식시킬 수 있는 방법이 있다.



[그림 66. Side Navigation 을 위한 힌트를 제공하는 뷰 2]

또한 위의 그림과 같이 메인 화면에서 사이드 네비게이션 기능을 알려주는 힌트를 제공한다.

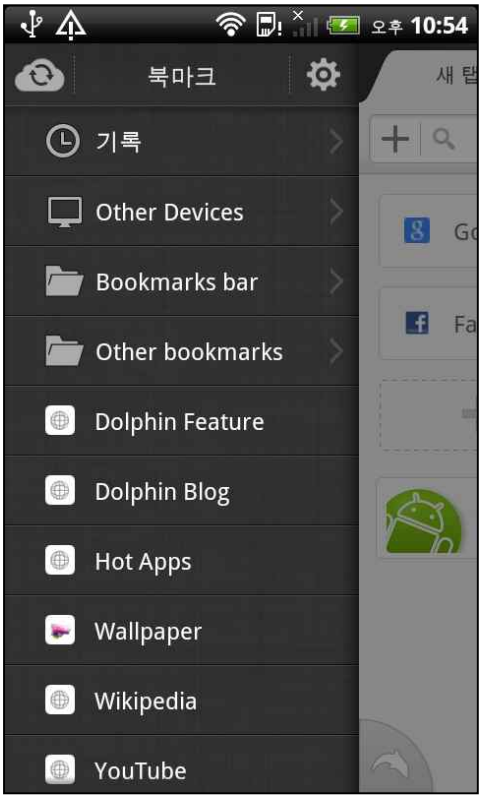
개선 방향 3) 현재 사용하고 있는 화면 표시



[그림 67. Side Navigationd 에서 현재 사용하고 있는 뷰를 알려주는 힌트 1]

사이드 네비게이션을 사용하는 대부분의 어플리케이션은 사용자가 현재 어떤 기능을 사용하는지에 대한 표시를 하지 않는다. 대표적으로 미국에서 가장 성공한 페이스북 어플리케이션만 봐도 현재 사용하고 있는 메뉴를 알려주지 않는다.
위의 그림과 같이, 화살표 또는 아이콘을 통하여 현재 사용하고 있는 기능을 알려준다면 사용자들에게 세부 정보를 알려 줄 수 있는 효과를 얻을 것이다.

개선 방향 4) 음영을 통한 사이드 메뉴 강조



[그림 68. Side Navigationd 에서 현재 사용하고 있는 뷰를 알려주는 힌트 2]

위의 그림과 같이 음영을 통해 사이드 네비게이션 메뉴를 강조하는 것은, 사용자에게 어느 부분에 포커싱되어 있는지에 대한 이해를 좀 더 직관적으로 제시해준다.

5) 사이드 네비게이션 구현 방법

아직까지 사이드 네비게이션 유아이는 안드로이드 플랫폼에서 제공되지 않지만, 페이스북, google+ 등 다양한 어플리케이션에서 사이드 네비게이션 유아이가 등장함에 따라 이에 관련된 오픈 소스의 종류는 매우 다양할 뿐만 아니라 인터넷 상에서도 쉽게 찾아볼 수 있다. 또한 안드로이드 플랫폼에서 사이드 네비게이션에 대한 개념이 정의되어 있지 않기 때문에 기능을 구현하는 방법은 오픈 소스마다 각각 다른 방법을 사용하고 있다.
사이드 네비게이션 관련 오픈 소스는 jar 파일로 배포되는 라이브러리가 아니라 소스코드로 배포되는 것을 확인할 수 있으며, 때문에 사이드 네비게이션 유아이를 개발자가 필요한 형식대로 커스텀이 가능하다.
아래에서는 오픈 소스의 코드를 분석하고, 사이드 네비게이션 기능을 구현하기 위한 간략한 방법을 소개하고자 한다.
1) https://github.com/johnkil/SideNavigation 을 통한 사이드 네비게이션 구현 방법

https://github.com/johnkil/SideNavigation 경로를 통해 사이드 네비게이션 기능을 구현하기 위한 라이브러리와 ActionBarSherlock, 샘플 코드를 다운로드 받는다. 이클립스에서 새로운 프로젝트를 생성한 후 위의 제공하는 두 개의 라이브러리를 등록한 후 사이드 네비게이션 기능을 구현한다.

```
<com.devspark.sidenavigation.SideNavigationView
    android:id="@+id/side_navigation_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

사이드 네비게이션을 통해 구현할 사이드 메뉴의 레이아웃은 com.devspark.sidenavigation.SideNavigationView 을 통하여 화면을 구성한다.

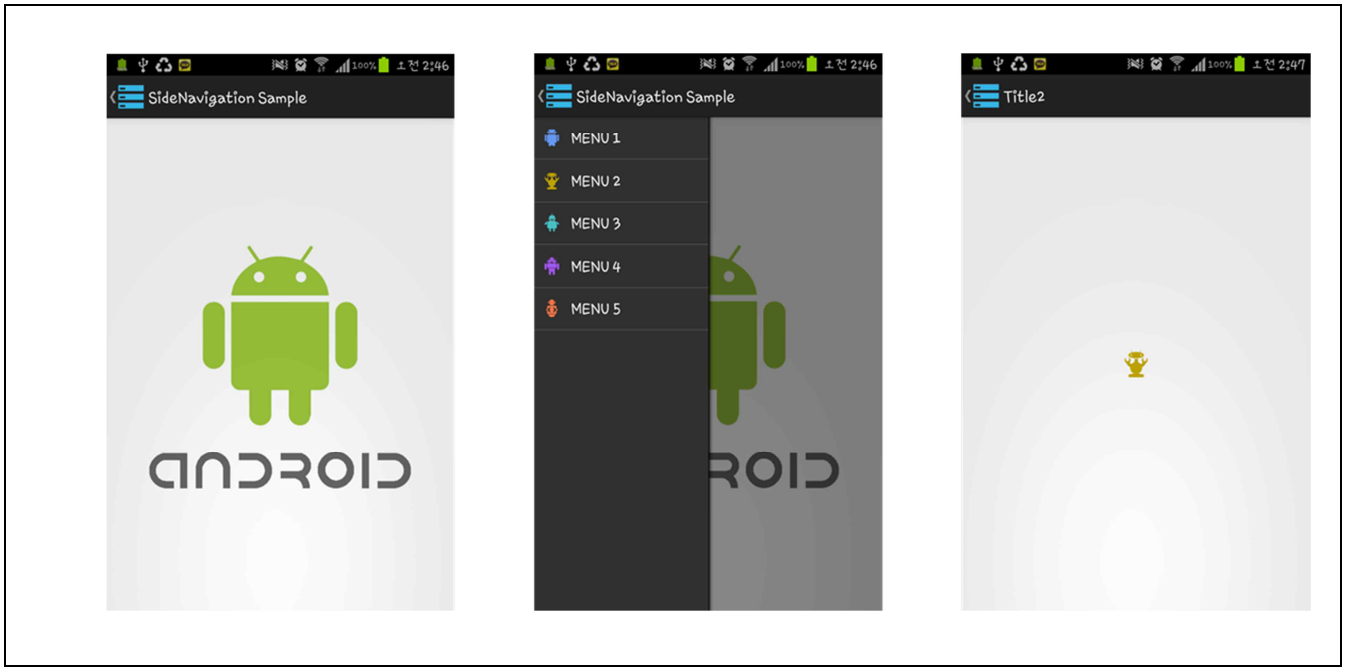
```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
        icon = (ImageView) findViewById(android.R.id.icon);
        sideNavigationView = (SideNavigationView)
findViewById(R.id.side_navigation_view);
        sideNavigationView.setMenuItems(R.menu.side_navigation_menu);
        sideNavigationView.setMenuClickCallback(this);

            . . . . . .

}
```

위의 예제 코드와 같이 onCreate 함수에서는 SideNavigationView 의 ID 를 선언한 후, setMenuItems 함수를 통해 구현하고자 하는 메뉴를 로드시킨다. 또한 setMenuClickCallback 함수를 통하여 각 메뉴를 눌렀을 때 발생할 이벤트를 연결한다.

```
@Override
public void onSideNavigationItemClick(int itemId) {
        switch (itemId) {
        case R.id.side_navigation_menu_item1:
// 메뉴를 선택하였을 때 발생할 이벤트 정의
                break;

        ......

        default:
                return;
        }
        finish();
}
```

onSideNavigationItemClick(int itemId) 함수는 메뉴 버튼을 눌렀을 때 발생하는 이벤트를 정의하기 위한 코드이다. 위의 코드는 SideNavigationView.setMenuClickCallback(this) 함수가 선언이 되어 있어야만 해당 함수로의 호출이 가능하다.

이 밖에도 사이드 네비게이션 라이브러리에는 아래와 같은 함수가 추가되어 있다.
toogleMenu() : 사이드 네비게이션 메뉴의 상태를 toogle 시키는 함수
showMenu() : 사이드 네비게이션 메뉴를 보여주는 함수
hideMenu() : 사이드 네비게이션 메뉴를 감추는 함수
isShown() : 사이드 네비게이션 메뉴가 화면에 보여지는지에 대해 알려주는 함수



[그림 69. Side Navigation 구현한 화면]

위의 예제 코드를 작성한 결과 UI 는 위의 그림과 같다. 왼쪽 상단의 버튼을 터치하면 메뉴 화면이 보여지거나 사라진다. 또한 메뉴의 각 기능을 선택하면, 그 기능에 해당하는 이벤트를 수행하는 것을 확인할 수 있다.

6) 사이드 네비게이션 관련 오픈 소스 예제 사이트

아직까지 사이드 네비게이션 유아이는 안드로이드 SDK 에 포함되지 않았기 때문에 위에서 소개한 오픈 소스 이외에도 사이드 네비게이션에 관련된 오픈 소스는 무수히 많다.
하지만 다양한 오픈 소스는 유아이를 구현하는 방법이 각기 다를 뿐만 아니라, 구현하는 방식에 따라 화면에 보여지는 화면도 다르므로, 상황에 따라 필요한 소스를 선택하여 사용하거나 커스텀하여 사용하면 된다.

https://github.com/johnkil/SideNavigation
https://github.com/korovyansk/android-fb-like-slideout-navigation
https://github.com/darvds/RibbonMenu
https://bitbucket.org/jfeinstein10/slidingmenu/overview
https://github.com/Gregadeaux/android-fly-in-app-navigation
https://github.com/gitgrimbo/android-sliding-menu-demo

[별첨10] Data Binding

1) 개요

안드로이드(Android) 어플리케이션(Application) 개발의 기본 단위는 액티비티(Activity)이다. 액티비티는 어플리케이션의 흐름을 제어하고, 사용자 입력을 다루며, 데이터(Data)/서비스(Service)레이어(Layer)와 통신 하는 등의 일을 한다. 이러한 안드로이드 SDK 의 디자인 방법은 액티비티에 과부화를 주며 꽤나 무겁다.

2) MVVM(Model View ViewModel)

A. 개요

MVVM 은 MVC 와 MVP 를 기반으로 한 아키텍쳐(Architecture) 패턴(Pattern)이다. 이 패턴은 어플리케이션 개발에서 UI(User Interface)와 비지니스-로직(Business-Logic) 그리고 행동을 더 명확히 나누고자 한 시도이며, 선언된 데이터 바인딩들을 이용해서 다른 레이어들로부터 뷰(View)들을 나눌 수 있도록 하는데, 이것은 UI 작업과 개발 작업을 같은 코드(Code) 베이스에서 거의 동시 작업을 가능케 한다.

B. 히스토리

MVVM 은 Microsoft 에서 처음 정의 되었으며, WPF(Windows Presentation Foundation)와 실버라이트(Silverlight)에서 같이 사용하기 위해 만들어졌다. 정식으로 소개 되어진 것은 2005 년 John Grossman 의 블로그에서 Avalon 에 대한 글로부터 나왔다.

C. 모델(Model)

다른 MV* 모델들과 같이, MVVM 에 있는 모델은 어플리케이션에서 사용될 도메인(Domain)데이터와 정보를 대신한다. 도메인 데이터의 대표적인 예는 사용자 계정(e.g. 이름, 아바타, 이메일) 또는 음반(e.g. 제목, 년도, 앨범) 등이 될 수 있다.
모델은 정보를 가지고만 있을 뿐 어떠한 행동도 하지 않는다. 정보 구성방식이나 화면에 어떻게 보여질지에 대한 조작을 하지 않으며, 이런 일들을 담당하지도 않는다. 대신 구성방식이나 조작은 View 에서 하게 될 것이고, 행동에 대한 것들은 비지니스-로직(Business Logic)을 다루는 일로 간주되어 다른 레이어에서 다루어진다. Model 과 소통하는 일도 여기서 행해진다.
한가지 예외가 있다면 Model 들이 데이터가 유효한지 확인하고 기존 모델을 업데이트 하는 경우이다. (e.g. e-mail 주소 형식 검증 등..)

D. 뷰(View)

MVC(Model View Controller)에서와 같이, 뷰(View)는 사용자와 어플리케이션이 유일하게 소통하는 부분이다. 뷰는 상호적인 UI 로 ViewModel 에 상태 정보를 제공하는데, 이런 측면에서 보면 MVVM 의 뷰는 수동적이기 보다는 능동적인 View 이다.

MVC 프레임워크에서의 뷰는 어플리케이션에서 쓰이는 모델들을 알지 못하며, 단지 컨트롤러(Controller)에 의해서 다루어지는 반면, MVVM 의 능동적인 뷰는 모델과 뷰모델(ViewModel)을 알기 위한 데이터 바인딩, 이벤트 그리고 행동들을 가지기 때문에, 계속적으로 뷰모델의 이벤트를 핸들링 할 수 있다.

E. 뷰모델(ViewModel)

뷰모델(ViewModel)은 특화된 컨트롤러로 생각할 수 있으며, 데이터 컨버터(Converter)로 작동한다. 모델 정보를 뷰 정보로 바꾸어 주며, 뷰에서 모델로 명령을 전달한다.

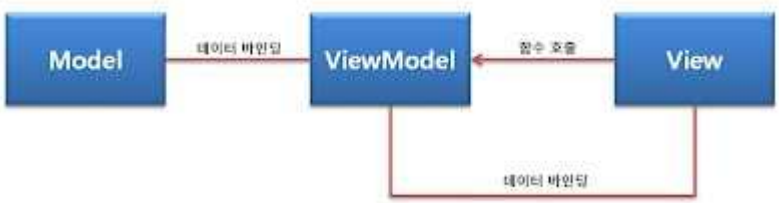
간단한 예로, 모델에 날짜 정보가 저장되어 있다고 생각해 보자. 모델은 사용자를 위해서 날짜 정보를 어떠한 포맷으로 바꾸어 주기 보다는, 간단히 데이터만 가지고 있는다. 그리고, 뷰에서는 날짜에 대한 디스플레이(Display) 포맷(Format)을 가지고 있는다. 그러면, 뷰모델에서는 모델에 있는 날짜 정보를 가지고 와서 뷰에 있는 디스플레이 포맷으로 바꾸어 주는 일을 하게 된다. 즉, 모델과 뷰의 중개자 역할을 하는 셈이다.

이러한 측면에서, 뷰모델은 뷰보다는 모델을 더 바라보고 있지만, View 대부분의 디스플레이 로직을 처리한다. 또한 뷰의 상태변화를 위한 메소드들을 가지고 있을 수 있어, 뷰에서 어떠한 동작이 있을 때, 모델을 갱신하고 뷰에 있는 이벤트를 발생시킨다.

간단히 말해서, 뷰모델은 UI 레이어 뒤에서 View 가 필요로 할 때 Model 에서 데이터를 가져와 보여준다.

F. MVVM

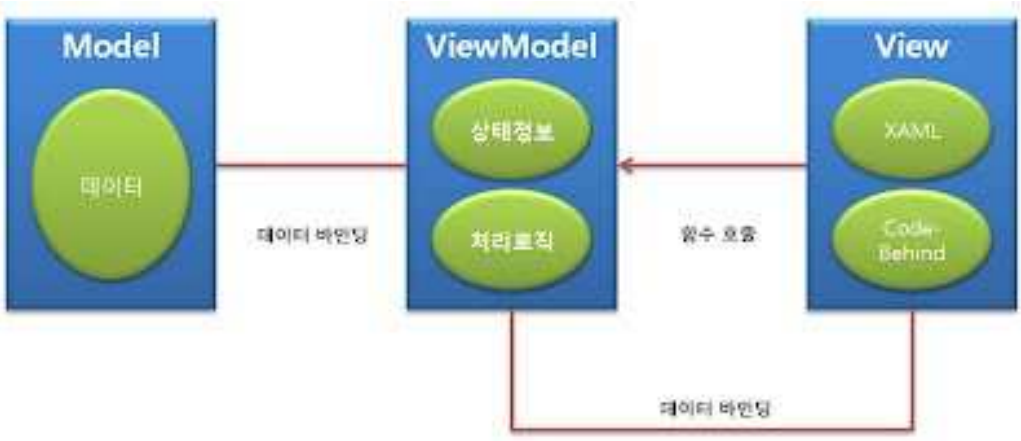
결론적으로 MVVM 은 아래 그림과 같이 사용자와의 상호통신이 활발하고 동적인 사용자 인터페이스를 가지는 어플리케이션에 적합하도록 만든 패턴이다.



[그림 70. MVVM 패턴]

MVVM 을 구성하는 각 컴포넌트가 담당하는 일을 정리해보면 다음과 같다

Model: 데이터와 그 처리 로직을 가진다.
View: 사용자의 입력정보를 수신하고, UI 요소를 그린다.
ViewModel: View 에 보여지는 데이터와 수정에 필요한 로직을 가진다.
MVVM 이 가장 활발히 사용되는 분야가 위에서 언급한 Microsoft 의 WPF 개발 분야인데 WPF 의 예를 구조화 하면 다음 그림과 같다.



[그림 71. MVVM 의 구조]

WPF 에서 UI 를 만들면 코드 비하인드(Code-Behind)와 XAML 이 쌍으로 생성된다. 일반적으로 XAML 에서는 각 디자인 요소에 이름을 주고, 그 이름에 따라 코드 비하인드에서 각종 처리 코드를 작성해야 한다. 이 경우, 추후에 디자인 요구사항이 변경되면 변경된 XAML 파일에 맞게 개발 코드를 일일이 수정해야 하는 문제가 있다.
MVVM 에서는 뷰를 XAML 와 코드 비하인드의 쌍으로 본다. 그리고 모델은 시스템에서 처리해야 할 데이터와 그 처리 로직을 의미한다. 이 뷰와 모델은 서로 독립적이며 연관성도 없다. 다만 중간에 뷰모델을 두어 모델의 데이터가 생성되거나, 변경 되었을 때 바인딩으로 엮여 있는 뷰의 값을 변경 시켜주는 역할을 한다. 뷰모델과 뷰는 바인딩 방식으로 데이터를 주고 받기 때문에, 뷰 모델은 뷰를 구성하고 있는 객체 이름과 독립적으로 만들어 질 수 있다.

3) Android-Binding

Android-Binding 프레임워크는 안드로이드가 제공하는 XML 레이아웃 뷰 바인딩 메카니즘을 위한 오픈소스 프레임워크이다. 뷰 위젯들과 백엔드 액티비티들을 분리시켜 안드로이드 어플리케이션의 개발을 도와주며, MVP 혹은 MVVM 패턴에서 최적으로 작동한다.

A. Hello World

Android-Binding 을 이용하기 위해서는 다음의 준비사항이 필요하다.

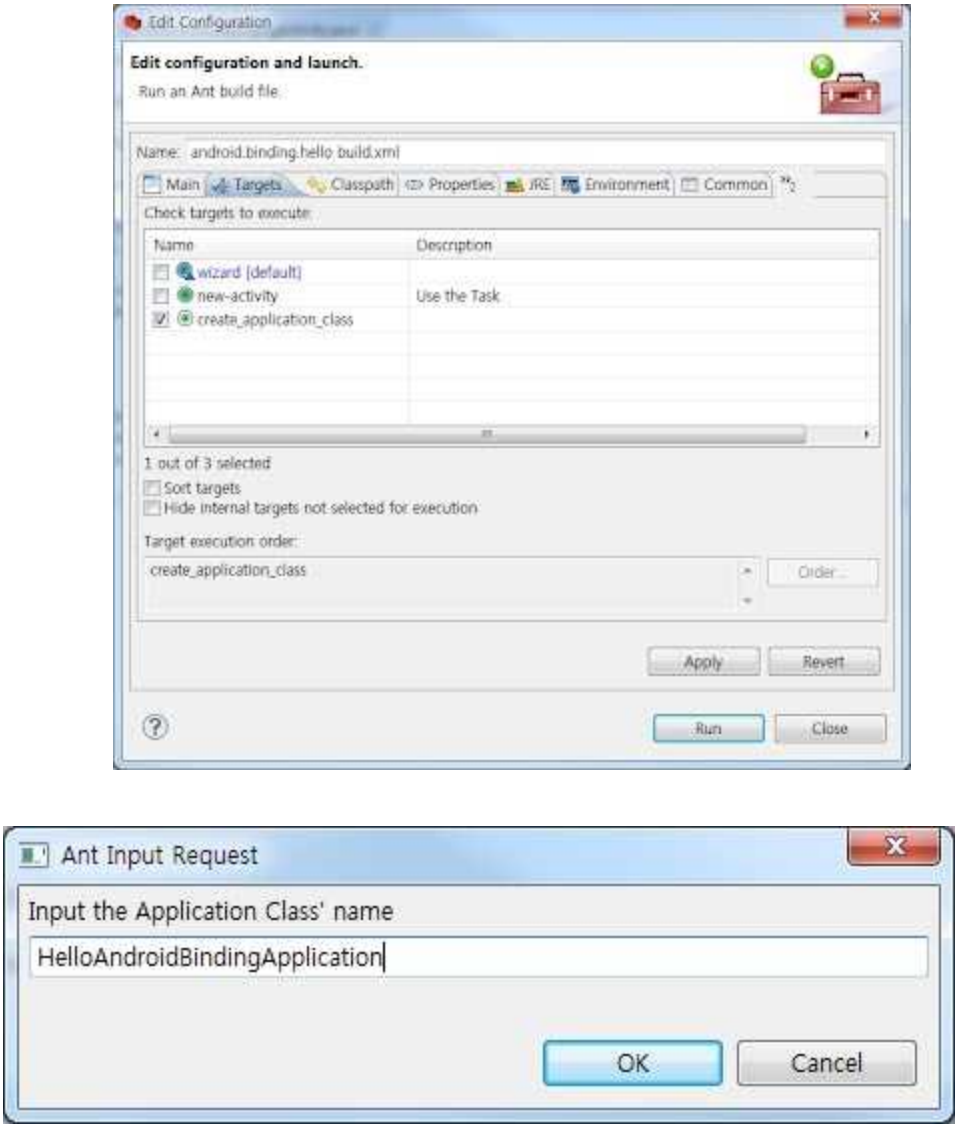
안드로이드 개발환경이 갖추어진 이클립스(Eclipse) IDE
최신 Android-Binding 라이브러리
Ant Build Package(앤트 빌드 패키지) - 선택사항

B. Android-Binding 설정

이클립스 IDE 에서 기본 안드로이드 프로젝트를 생성하고,프로젝트의 "libs" 폴더에 android-binding library 를 설치 한다. 앤트 빌드 패키지까지 설치 하였다면 다음 그림과 같이 라이브러리 jar 파일들과 앤트 빌드를 위한 build.xml 을 볼 수 있다.

[그림 72. Binding 프레임워크의 설정화면]

Android-Binding 초기화를 위해 어플리케이션 파일이 필요한데, build.xml 을 이용하여 쉽게 생성할 수 있다. build.xml 을 "create_application_class"를 타겟으로 앤트 빌드를 실행하면, 잠시 후 생성할 어플리케이션 클래스 이름을 입력하는 프롬프트가 보이고, 원하는 이름을 입력 후 계속 진행하면 된다.



[그림 73. Ant 설정]

아무런 문제가 없다면 콘솔 창에서 "BUILD SUCCESSFUL" 메시지를 확인할 수 있다. 그리고는 어플리케이션 클래스에서는 바인딩 엔진을 초기화 하는 것이 필요한데, 다음과 같은 BinderV30.init() 코드가 해결해 준다.

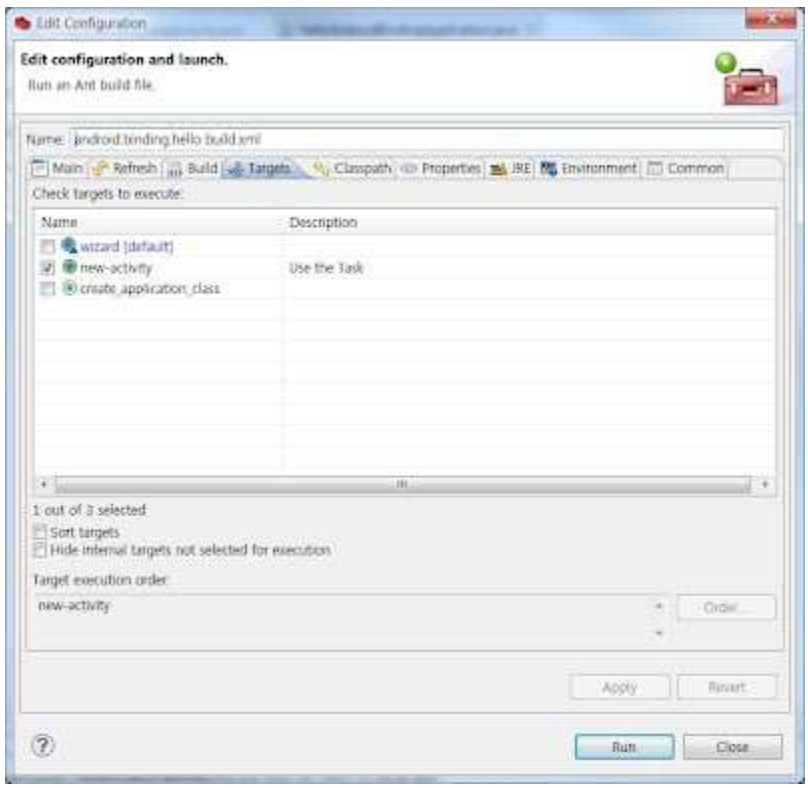
```
import gueei.binding.v30.BinderV30;
import android.app.Application;
public class HelloAndroidBindingApplication extends Application {
@Override
public void onCreate() {
super.onCreate();
```

```
BinderV30.init(this);
}
}
```

BinderV30.init(Application) 메서드는 어플리케이션 라이프 사이클(Life-Cycle) 전체에 걸쳐 단 한번만 실행 되어야 하며, 바인딩을 등록하고 마크업과 커스텀 뷰 클래스들을 지원할 수 있는 바인딩 프로바이더(Provider)들을 초기화 한다.

C. 액티비티 클래스 생성

Android-Binding 을 이용하여 어플리케이션 클래스를 생성하였다면, 이제는 액티비티 클래스와 관련 리소스(Resource) 파일들을 생성하여야 한다. 어플리케이션 클래스와 마찬가지로 이것 또한 앤트 빌드를 이용하여 생성 할 수 있다. build.xml 을 "new-activity"를 타겟으로 앤트 빌드를 실행하면,잠시 후 생성할 액티비티 클래스 이름을 입력하는 프롬프트가 보이고, 원하는 이름을 입력 후 계속 진행하면 된다.



[그림 74 액티비티 클래스 설정]

아무런 문제가 없다면 잠시 후 콘솔 창에서 "BUILD SUCCESSFUL" 메시지를 확인할 수 있고, 프로젝트를 리프레쉬 하면 클래스와 관련 리소스들이 생성된 것을 볼 수 있다.

```
import gueei.binding.Command;
import gueei.binding.observables.StringObservable;
import gueei.binding.v30.app.BindingActivityV30;
import android.os.Bundle;
import android.view.View;
public class HelloAndroidBindingActivity extends BindingActivityV30 {
public final StringObservable HelloWorld =
new StringObservable("Hello world from Android-Binding");
public final Command SayGoodBye =
new Command(){
@Override
public void Invoke(View arg0, Object... arg1) {
HelloWorld.set("Good bye :)");
}
};
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
this.inflateAndBind(R.xml.helloandroidbindingactivity_metadata, this);
}
}
```

1. xml/{name}_metadata.xml: 바인딩 액티비티를 위한 메타데이터 파일.

2. xml-v14/{name}_metadata.xml: 액션 바를 위한 메타데이터 파일. 오직 SDK14+에서 읽기 가능하다.
3. layout/{name}_layout.xml: 메인 레이아웃 파일
4. menu/{name}_optionsmenu.xml: 옵션 메뉴 템플릿
5. src/{name}.java: 생성된 액티비티 클래스

4) 기본 안드로이드 SDK 를 사용한 리스트 뷰

안드로이드 어플리케이션 개발에서 리스트뷰(ListView)는 가장 많이 사용 되는 위젯 중 하나이지만, 아이템 소스를 가지고 있는 리스트(List)에서 데이터를 가져와 표시하기 위한 작업은 꽤나 번거로운 작업이다.
안드로이드 SDK 에서 제공되는 SimpleAdapter 의 경우 대부분의 상황에서 그다지 유용하지 않아서, 개발자가 아이템 소스와 리스트뷰를 위한 어댑터(Adapter)를 구현하는데 대부분의 시간을 들여야 하기 때문이다.
이에 본 문서에서는 기본 안드로이드 SDK 가 제공하는 기능으로 리스트뷰를 구현하는 것을 다루고, 이것을 Android-Binding 을 이용하여 어떻게 다룰 수 있는지 살펴보겠다.

A. 모델(Model)

먼저, 리스트뷰에 보여질 모델 데이터를 위해서 안드로이드 표준 리소스 포맷으로 /res/values/string.xml 에 다음과 같이 정의 하였다.

```
<?xml version="1.0" encoding="utf-8"standalone="no"?>
<resources>
...
<string-array name="flowerNames">
<item>갈퀴꼭두서니</item>
<item>개구리자리</item>
<item>개망초</item>
<item>개쇠스랑개비</item>
<item>개오동</item>
<item>개옻나무</item>
<item>갯취</item>
...
</string-array>
...
</resources>
```

이렇게 정의한 리소스는 액티비티 클래스에서 Context.getResources()로 접근하여 얻어온다. 본 예제에서는 기본 스트링(String) 클래스를 모델 클래스로 사용하며 데이터는 그 자체가 된다.

```
getResources().getStringArray(R.array.flowerNames);
```

B. 뷰(View)

모델 데이터를 사용자에게 보여주기 위한 수단으로 기본 리스트뷰(ListView)를 사용하며, 기본 리스트뷰는 그다지 특별할 것 없이 다음과 같이 안드로이드 표준 XML 포맷으로 정의한다.

```
<ListView
android:id="@+id/listView"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
/>
```

C. 어댑터(Adapter)

어댑터(Adapter)는 리스트 객체를 리스트뷰에 표시해 주는 기능을 담당한다. 리스트 객체의 내용과 리스트 항목의 레이아웃을 연결 시켜주는 역할을 하는 것이다. 본 예제에서는 리소스로 얻어온 데이터를 넘겨받아 사용자가 원하는 레이아웃으로 표시가 위해서 사용자 정의 어댑터를 만들었다.

```
import android.content.Context;
import android.graphics.Color;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
public class DemoListViewAdapter extends BaseAdapter{
private Context context;
private String [] flowerNames;
public DemoListViewAdapter(Context context, String [] flowerNames){
this.context = context;
this.flowerNames = flowerNames;
}
@Override
```

```
public int getCount() {
return flowerNames.length;
}
@Override
public View getView(int position, View view, ViewGroup viewGroup) {
if(view == null){
TextView textView = new TextView(this.context);
textView.setText(flowerNames[position]);
textView.setTextSize(25f);
textView.setTextColor(Color.BLACK);
return textView;
}else{
TextView textView = (TextView) view;
textView.setText(flowerNames[position]);
return textView;
}
}
@Override
public Object getItem(int position) {
return flowerNames[position];
}
@Override
public long getItemId(int position) {
return 0;
}
}
```

D. 행위(Behavior)

사용자에 의해 리스트뷰에서 어떠한 이벤트(Event)가 발생 하였을 때, 그 것을 감지하여 해당 이벤트에 대한 행위가 이루어져야 하는데, 표준 안드로이드 SDK 에서는 이를 위해 이벤트 리스너(Listener)를 사용한다. 다음은 리스트 뷰에서 아이템에 대해 클릭 이벤트가 발생하였을 때, 작동해야 하는 행위를 나타내고 있는 리스너 이다.

```
import android.content.Context;
import android.view.View;
```

```
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListAdapter;
import android.widget.Toast;
public class DemoListItemClickListener implements OnItemClickListener {
private Context context;
private ListAdapter listAdapter;
public DemoListItemClickListener(Context context,
ListAdapter listAdapter){
this.context = context;
this.listAdapter = listAdapter;
}
@Override
public void onItemClick(AdapterView<?> arg0,
View view, int position, long id) {
Toast.makeText(context,
"Selected Item : " + listAdapter.getItem(position),
Toast.LENGTH_SHORT).show();
}
}
```

해당 이벤트에 대해서 감시를 하고 있는 리스너는 리스트뷰에 대한 어뎁터를 전달받아서 어떤 아이템에서 이벤트가 발생 하였을 때 어뎁터를 이용하여 모델에 대한 데이터 정보를 가져와 활용한다.

E. 액티비티(Activity)

액티비티는 어플리케이션의 흐름을 제어하고, 사용자의 입력을 다루며, 데이터 혹은 다른 서비스 레이어와 통신을 하는 등의 처리를 하게 되는데, 위에서 정의한 모델(Model), 뷰(View), 어뎁터(Adaptor), 행위(Behavior) 들을 액티비티 내에서 모두 다루어야 한다.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.ListView;
public class MainActivity extends Activity {
@Override
```

```
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
String [] flowerNames =
getResources().getStringArray(R.array.flowerNames);
DemoListViewAdapter demoListViewAdapter =
new DemoListViewAdapter(this, flowerNames);
DemoListItemClickListener demoListItemClickListener =
new DemoListItemClickListener(
getApplicationContext(), demoListViewAdapter);
ListView listView = (ListView) this.findViewById(R.id.listView);
listView.setAdapter(demoListViewAdapter);
listView.setOnItemClickListener(demoListItemClickListener);
}
}
```

액티비티는 액티비티가 생성되는 시점에 리소스에서 모델 데이터를 읽어 들이고, 읽어 들여진 모델 데이터를 어뎁터에 전달한 후, 해당 어뎁터를 이벤트 리스너에게 넘겨 준다. 그리고 리스트뷰를 위한 레이아웃 구성도 액티비티가 하게 된다.

5) Android-Binding 을 사용한 리스트 뷰

A. 모델(Model)

리스트뷰에 보여질 모델 데이터는 "안드로이드 SDK 를 사용한 리스트 뷰" 에서와 같이 안드로이드 표준 리소스 포맷으로 /res/values/string.xml 에 정의 하였고, 액티비티 클래스에서 Context.getResources()로 접근하여 얻어온다.

B. 뷰모델(ViewModel)

뷰모델은 커스텀 네임스페이스를 사용하여 리스트를 위한 아이템소스(Item Source)를 연결하고 아이템 템플릿(Item Template)을 지정하며, 이벤트에 대한 이벤트 처리 객체를 지정한다.

```
public ArrayListObservable<String> flowerItems =
new ArrayListObservable<String>(String.class);
public final Command showName = new Command(){
public void Invoke(View view, Object... args) {
Toast.makeText(
```

```
view.getContext(),
flowerItems.getItem((Integer)args[1]),
Toast.LENGTH_SHORT).show();
}
};
```

```
…
xmlns:binding="http://www.gueei.com/android-binding/"
…
binding:itemSource="flowerItems"
binding:itemTemplate="@layout/arraylist_item"
binding:onItemClicked="showName"
…
```

binding:itemSource="flowerItems"
아이템소스는 리스트뷰가 디스플레이 하는 데이터 컬렉션(Collection)인데, 현재 Android-Binding 에는 리스트뷰에 바인드 할 수 있는 컬렉션 타입이 두 가지가 있다. 하나는 CursorCollection 이고 다른 하나는 본 예제에서 다루고 있는 ArrayListObservable 이다. flowerItems 라는 이름으로 생성된 ArrayListObservable 객체를 바인딩 한다.
binding:onItemClicked="showName"
리스트뷰에서 이벤트가 발생했을 때, 해당 이벤트를 처리하는 객체를 지정한다.
binding:itemTemplate="@layout/arraylist_item"
아이템 템플릿(Item Template)은 리스트뷰에 보이는 각각의 객체들이 어떻게 보여져야 하는지에 대해 정의를 하고 있는 다른 레이아웃 아이디(Id)에 대한 레퍼런스 이다.

C. 뷰(View)

뷰와 관련된 디스플레이 로직의 모든 것은 XML 파일에 있고, 단지 이것만 보면 된다. 아래 두 개의 XML 코드는 정확히 레이아웃 XML 로서 작동한다.

```
[res/layout/demomainactivity_layout.xml]
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:binding="http://www.gueei.com/android-binding/"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
```

```
android:orientation="vertical">
<ListView
android:id="@+id/listView"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
binding:itemSource="flowerItems"
binding:itemTemplate="@layout/arraylist_item"
binding:onItemClicked="showName" />
</LinearLayout>
```

```
[res/layout/arraylist_item.xml]
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:binding="http://www.gueei.com/android-binding/"
android:id="@android:id/text1"
android:layout_width="fill_parent"
android:layout_height="?android:attr/listPreferredItemHeight"
android:gravity="center_vertical"
android:paddingLeft="6dip"
android:textAppearance="?android:attr/textAppearanceLarge"
binding:text="."
/>
```

본 예제에서는 TextView 의 Text 를 "."으로 바인드 했고 이것은 객체 자신을 의미하는데, 현재 리스트의 모델은 String 클래스이기 때문에 String 의 문자열 값을 디스플레이한다.

D. 액티비티(Activity)

액티비티는 어플리케이션의 흐름을 제어하고, 사용자의 입력을 다루며, 데이터 혹은 다른 서비스 레이어와 통신을 하는 등의 처리를 하게 되는데, 위에서 정의한 모델(Model), 뷰(View), 뷰모델(ViewModel) 들을 액티비티 내에서 다루어야 한다.

```
import gueei.binding.Command;
import gueei.binding.collections.ArrayListObservable;
import gueei.binding.v30.app.BindingActivityV30;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Toast;
public class DemoMainActivity extends BindingActivityV30 {
public final ArrayListObservable<String> flowerItems =
new ArrayListObservable<String>(String.class);
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
flowerItems.setArray(
getResources().getStringArray(R.array.flowerNames));
this.inflateAndBind(R.xml.demomainactivity_metadata, this);
}
public final Command showName = new Command(){
public void Invoke(View view, Object... args) {
Toast.makeText(
view.getContext(),
flowerItems.getItem((Integer)args[1]),
Toast.LENGTH_SHORT).show();
}
};
}
```

위 코드에서 보듯 액티비티는 단지 생성되는 시점에 리소스에서 모델 데이터를 읽어 들이고, 읽어 들여진 모델 데이터를 리스트뷰에 바인드 할 수 있는 ArrayListObservable 클래스의 객체에 전달하는 일만 한다.
리스트뷰에 데이터 소스를 연결시키기 위한 어떤 어댑터도 필요 없으며, 사용자의 이벤트를 감지하기 위한 어떠한 이벤트 리스너도 있을 이유가 없다. 모든 것은 Android-Binding 이 해준다.

6) Android-Binding 을 사용한 리스트 뷰 - 사용자 정의 뷰모델

위 예제에서는 기본 스트링(String) 클래스를 뷰모델로 사용하여 리스트뷰에 나열하고, 해당 아이템 클릭 시 단지 이름만을 보이는 간단한 형태의 예를 보았다. 이번에는 사용자 정의 클래스를 뷰모델로 사용해서 다루어본다.

A. 모델(Model)

위에서와 같이 모델 데이터는 /res/values/string.xml 에 정의 하였고, 액티비티 클래스에서 Context.getResources()로 접근하여 얻어온다.

```
<?xml version="1.0" encoding="utf-8"standalone="no"?>
<resources>
...
<string-array name="flowers">
<item>갈퀴꼭두서니,6800</item>
<item>개구리자리,1900</item>
<item>개망초,6400</item>
<item>개쇠스랑개비,9500</item>
<item>개오동,100</item>
<item>개옻나무,2700</item>
<item>갯취,10000</item>
<item>거북꼬리,6300</item>
<item>겹삼잎국화,8200</item>
<item>계요등,9800</item>
...
</string-array>
...
</resources>
```

콤마(Comma)를 구분자로 왼쪽에는 아이템의 이름 그리고 오른쪽에는 아이템의 가격정보를 정의 하였다. 해당 데이터에 매핑되는 모델 객체는 다음과 같다.

```
public class FlowerModel {
private String name;
private Long price;
public FlowerModel(){}
public FlowerModel(String str){
String [] temp = str.split(",");
this.name = temp[0];
this.price = Long.parseLong(temp[1]);
}
public FlowerModel(String name, Long price){
this.name = name;
this.price = price;
```

```
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public Long getPrice() {
return price == null ? 0 : price;
}
public void setPrice(Long price) {
this.price = price;
}
}
```

B. 뷰모델(ViewModel)

뷰모델은 모델 객체를 받아 어떤 데이터를 디스플레이 할 것인지를 결정하고, 이벤트 처리객체를 지정한다.

리스트뷰에 디스플레이 되는 아이템 소스는 ArrayListObservable 을 상속받아 지정된 뷰모델에 대한 데이터 컬렉션을 만들어 바인딩 한다. 본 예제에서는 flowerItemSource 라는 이름으로 지정되었다.

```
FlowerItemSource flowerItemSource = new FlowerItemSource(this);
...
import gueei.binding.collections.ArrayListObservable;
import java.util.ArrayList;
import java.util.Collection;
import android.content.Context;
public class FlowerItemSource extends ArrayListObservable<FlowerViewModel> {
public FlowerItemSource(Context context){
super(FlowerViewModel.class);
this.setAll(getFlowerViewModels(context));
}
private Collection<FlowerViewModel> getFlowerViewModels(Context context){
String [] flowers =
```

```
context.getResources().getStringArray(
R.array.flowers);
Collection<FlowerViewModel> collection =
new ArrayList<FlowerViewModel>(flowers.length);
for(String flower : flowers){
collection.add(new FlowerViewModel(
new FlowerModel(flower)));
}
return collection;
}
}
```

```
import gueei.binding.Command;
import gueei.binding.DependentObservable;
import android.view.View;
import android.widget.Toast;
public class FlowerViewModel {
private FlowerModel model;
public FlowerViewModel(FlowerModel model){
this.model = model;
}
public final Command showFlowerName = new Command(){
private static final String MESSAGE_FORMAT =
"name : %s, price : %,05d";
public void Invoke(View view, Object... args) {
Toast.makeText(view.getContext(),
String.format(MESSAGE_FORMAT, model.getName(),
model.getPrice()),
Toast.LENGTH_SHORT).show();
}
};
public final DependentObservable<String> flowerName =
new DependentObservable<String>(String.class){
@Override
```

```
public String calculateValue(Object... args)
throws Exception {
return model.getName();
}
};
}
```

```
...
xmlns:binding="http://www.gueei.com/android-binding/"
...
binding:itemSource="flowerItemSource"
binding:itemTemplate="@layout/arraylist_item"
...
binding:text="flowerName"
binding:onClick="showFlowerName"
...
```

C. 뷰(View)

뷰와 관련된 디스플레이 모든 로직은 다음 두 개의 XML 파일에 있다.

```
[res/layout/demomainactivity_layout.xml]
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:binding="http://www.gueei.com/android-binding/"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<ListView
android:id="@+id/listView"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
binding:itemSource="flowerItemSource"
binding:itemTemplate="@layout/arraylist_item"
/>
</LinearLayout>
```

```
[res/layout/arraylist_item.xml]
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:binding="http://www.gueei.com/android-binding/"
android:id="@android:id/text1"
android:layout_width="fill_parent"
android:layout_height="?android:attr/listPreferredItemHeight"
android:gravity="center_vertical"
android:paddingLeft="6dip"
android:textAppearance="?android:attr/textAppearanceLarge"
binding:text="flowerName"
binding:onClick="showFlowerName"
/>
```

TextView 의 text 를 "flowerName"으로 바인드 했고, 이것은 뷰모델에서 정의된 DependentObservable 의 객체로 해당 아이템 모델의 getName() 메서드를 호출하여 그 값을 디스플레이한다.

D. 액티비티(Activity)

액티비티에서는 단지 위에서 다루어진 아이템 소스만 처리하고, 나머지 모든 것은 Android-Binding 이 처리한다.

```
import gueei.binding.v30.app.BindingActivityV30;
import android.os.Bundle;
public class DemoMainActivity extends BindingActivityV30 {
public FlowerItemSource flowerItemSource;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
flowerItemSource = new FlowerItemSource(this);
this.inflateAndBind(R.xml.demomainactivity_metadata, this);
}
}
```

7) Android-Binding 을 사용한 리스트 뷰 - 커서(Cursor)

지금까지 정적인 리스트 정보들을 리스트 뷰에 바인딩 하는 것을 다루었는데, 이번에는 안드로이드에서 중요한 개념 중 하나인 커서(Cursor)를 사용한 동적 리스트 정보들의 바인딩에 대해 살펴보겠다.
커서의 컨텐츠(Contents)는 실제로 그것이 필요하기 전까지는 메모리를 사용하지 않는다. 그래서 1000 개의 레코드(Record)를 반환하는 쿼리(Query)를 수행 한다 하여도, 1000 개의 레코드를 처리하는 오버헤드와 메모리가 필요하지 않다. 단지 화면에 보여질 10 개에서 20 개의 레코드만이 메모리를 사용할 것이며, 상대적으로 적은 양의 메모리를 사용하는 것을 고려하면 커서는 객체의 정적인 컬렉션 보다는 나은 대안일 것이다.
Android-Binding 에서는 커서를 뷰에 바인딩 하기 위해서 CursorCollection 을 사용한다. CursorCollection 의 사용법은 ArrayListObservable 과 매우 유사하며, 커서의 아이템에게 뷰모델을 알리기 위한 심플한 생성자를 가진다.

A. 모델(Model)

본 예제에서는 SMS 수신 박스의 SMS 데이터를 사용하여 리스트뷰에 디스플레이 한다. SMS 데이터를 사용하기 위해서 Uri 클래스와 해당 데이터 파라미터 조합을 이용하여 질의를 해서 결과 레코드에 대한 커서를 얻어온다.

```
Uri uri = Uri.parse("content://sms/inbox");
String[] param = {"_id", "address", "date", "body"};
Cursor c = getContentResolver().query(uri, param, null, null, null);
```

모델 데이터로 사용될 필드는 아이디, 발신자, 날짜, 내용이 사용되며, 해당 데이터는 로우 모델(Row Model)이라는 뷰모델로 매핑되어 사용된다.

B. 로우 모델(Row Model)

CursorCollection 을 사용하기 위해서는 로우 모델의 정의가 필요한데,로우 모델은 반드시 IRowModel 인터페이스(Interface)혹은 RowModel 추상클래스의 서브클래스 이어야 하며,CursorCollection 을 생성하는 시점에 뷰모델의 타입을 알리기 위해 로우 모델 클래스를 이용한다.

```
import gueei.binding.cursor.IdField;
import gueei.binding.cursor.RowModel;
import gueei.binding.cursor.StringField;
import java.text.SimpleDateFormat;
import java.util.Date;
public class MessageModel extends RowModel {
private static final SimpleDateFormat sdf =
```

```
new SimpleDateFormat("yyyy-MM-dd(E) HH:mm:ss");
public final IdField id = new IdField("_id");
public final StringField address = new StringField("address");
public final StringField date = new StringField("date");
public final StringField body = new StringField("body");
@Override
public void onInitialize() {
super.onInitialize();
date.set(sdf.format(new Date(Long.parseLong(date.get()))));
}
}
```

MessageModel 클래스는 ORM 프레임워크와 매우 비슷하다. Android-Binding 은 커서 레코드들을 로우 모델 객체로 변환한다.

위 예제에서 보듯 아이디, 발신자, 날짜, 내용 4 개의 컬럼을 가지고 있으며,이 컬럼들은 아래의 쿼리에서 사용된다. 로우가 읽혀지는 각각의 시간에 이 4 개의 값은 자동으로 로우 모델에 채워질 것이다.

```
String[] param = {"_id", "address", "date", "body"};
Cursor c = getContentResolver().query(uri, param, null, null, null);
```

C. 뷰모델 (ViewModel)

정의한 로우 모델 클래스를 이용하는 CursorCollection 객체를 아이템 소스로 사용하기 위해 생성하고, 이벤트를 핸들링 하기 위한 이벤트 처리 객체들을 생성한다.

```
public final CursorCollection<MessageModel> messages =
new CursorCollection<MessageModel>(MessageModel.class);
public final Observable<Object> messageClickedItem =
new Observable<Object>(Object.class);
public final Command showMessageBody = new Command(){
public void Invoke(View view, Object... args) {
Toast.makeText(view.getContext(),
((MessageModel)messageClickedItem.get()).body.get(),
Toast.LENGTH_SHORT).show();
}
};
```

```
…
xmlns:binding="http://www.gueei.com/android-binding/"
…
binding:itemSource="messages"
binding:itemTemplate="@layout/message_item"
binding:onItemClicked="showMessageBody"
binding:clickedItem="messageClickedItem"
…
binding:text="HTML(address, '&lt;br&gt;', body, '&lt;br&gt;', date)"
…
```

사용자 클릭 시 이벤트를 처리 하기 위해서 현재 이벤트가 발생된 아이템을 Observable 객체에 담아두고, 이벤트 처리 객체에서 이를 사용한다.

D. 뷰(View)

뷰와 관련된 디스플레이 모든 로직은 다음 두 개의 XML 파일에 있다.

```
[res/layout/demomainactivity_layout.xml]
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:binding="http://www.gueei.com/android-binding/"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical">
<ListView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
binding:itemSource="messages"
binding:itemTemplate="@layout/message_item"
binding:onItemClicked="showMessageBody"
binding:clickedItem="messageClickedItem"
/>
</LinearLayout>
```

```
[res/layout/arraylist_item.xml]
<?xml version="1.0" encoding="utf-8"?>
```

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:binding="http://www.gueei.com/android-binding/"
android:id="@android:id/text1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceLarge"
android:gravity="center_vertical"
android:paddingLeft="6dip"
android:minHeight="?android:attr/listPreferredItemHeight"
binding:text="HTML(address, '&lt;br&gt;', body, '&lt;br&gt;', date)"
/>
```

TextView 의 text 를 디스플레이 하기 위해서 이번에는 HTML() 을 사용하였고, 로우 모델의 필드값을 이용하여 데이터 정보를 디스플레이 한다.

E. 액티비티 (Activity)

액티비티에서는 SMS 수신 박스의 SMS 데이터를 사용하기 위해 질의를 실행하고 결과 커서를 CursorCollection 에 셋팅하는 작업만을 수행하고, 나머지 모델, 뷰, 뷰모델에 대한 모든 작업은 Android-Binding 이 처리한다.

```
import gueei.binding.Command;
import gueei.binding.Observable;
import gueei.binding.collections.CursorCollection;
import gueei.binding.v30.app.BindingActivityV30;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
public class DemoMainActivity extends BindingActivityV30 {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
init();
this.inflateAndBind(R.xml.demomainactivity_metadata, this);
}
```

```
private void init(){
Uri uri = Uri.parse("content://sms/inbox");
String[] param = {"_id", "address", "date", "body"};
Cursor c = getContentResolver().query(
uri, param, null, null, null);
messages.setCursor(c);
}
public final CursorCollection<MessageModel> messages =
new CursorCollection<MessageModel>(MessageModel.class);
public final Observable<Object> messageClickedItem =
new Observable<Object>(Object.class);
public final Command showMessageBody = new Command(){
public void Invoke(View view, Object... args) {
Toast.makeText(view.getContext(),
((MessageModel)messageClickedItem.get())
.body.get(),
Toast.LENGTH_SHORT).show();
}
};
}
```

참고자료

http://andytsui.wordpress.com
http://code.google.com/p/android-binding
http://msdn.microsoft.com/ko-kr/magazine/dd419663.aspx
http://addyosmani.com/blog/understanding-mvvm-a-guide-for-javascript-developers/