SmartDroid

Automation Application for Android

By: Ran Haveshush

Lior Ginsberg

Supervised by: Dr. Avivit Levy

Submitted as part of the requirements for B.Sc. in the software engineering department of Shenkar College

Thanks

First we like to thank Dr. Levy Avivit for guiding us through the hole project's process in all aspects, from documents and documentation, to project management and fitting the project's schedule.

We would also like to thank Shenkar's Software Engineering Department for suppling us all the needed instrumetns in order to test and run our application project.

Summary

SmartDroid is a mobile automation application that allows the user to set automatic operations on his device, instead of setting them manually. The application developed and designed to give the user the best experience while dealing with the phone's most complicated abilities in an easy manner and by that giving a new meaning to the phrase "SmartPhone".

TABLE OF CONTENT

1. Table of figures

2. Glossary of Terms

- **Action** – An Action is any type of operation that can be done by the smart phone manually.

- **Activity** - An Activity is a single thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI. While activities are often presented to the user as full-screen windows, they can also be used as floating windows, or background activities.

- **API Level** – **A**pplication **P**rogramming **I**nterface that is part of android open source project for developers. The android developing team, releases new version of the API as soon as a new android version is released.

- **BroadcastReciever** - Class for code that will receive intents sent by sendBroadcast() function.

- **CRUD** - Create, Read, Update and Delete operations on a database.

- **Fragment** - Fragment class is used to better modularize the code, build more sophisticated user interfaces for larger screens, and help scale their application between small and large screens. Fragments are like reusable component that can be attached or detach form activity.

- **Intent** - Intent is an abstract description of an operation to be performed. It is basically a passive data structure holding an abstract description of an action to be performed.

- **IntentService** - IntentService is a base class for services that handles asynchronous requests (expressed as Intents) on demand. Clients send requests through startService(Intent) calls. The service is started as needed and handles incoming intents by their time they were sent to the service.

- **OODBMS** - Object Oriented Database Management System.

- **RDBMS** - Relational Database Management System.

- **Rule** – A Rule is a container that's let you refer to a set of Triggers and Actions as a single unit. Each rule when created gets a unique identifier in order to manipulate it and keep track on. The Rule have a special value that indicate his current state, whether it is satisfied or not. When a Rule is satisfied that's mean that all of his triggers is satisfied. The Rule can be activated or deactivated.

- **Service** - A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use.

- **Trigger** – A Trigger is type of event that thrown by the operating system, which is being handle by special class of the Android API called BroadcastReciever.

3. Introduction

3.1 Quick Overview

SmartDroid is an automation application for android that basically takes the android device capabilities of environment and state awareness and matches the desired state according to the user pre-definition rules and preform the specified actions for the user in an automatic manner.

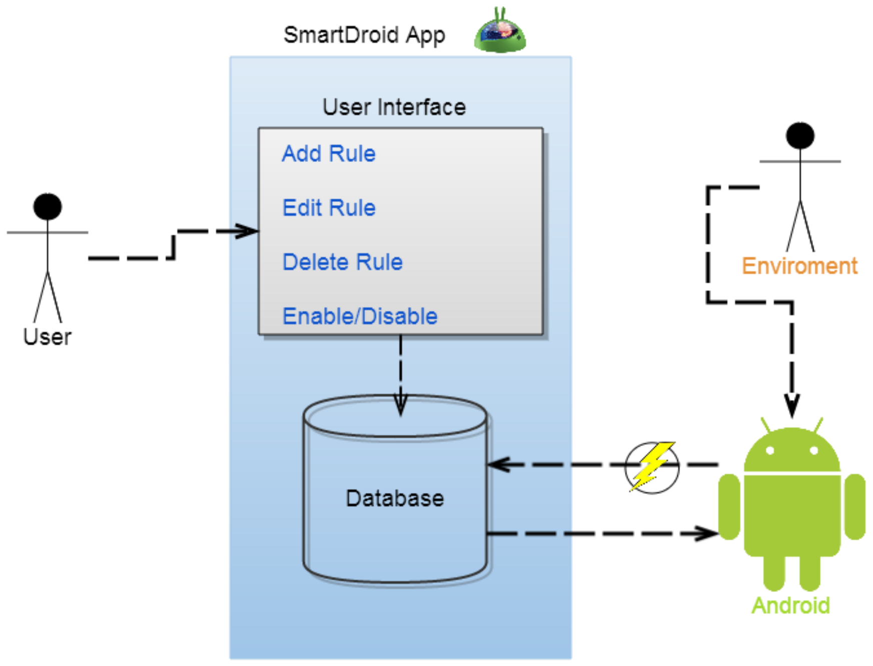


Figure 1 - System Overview Diagram.

In this overview diagram we can see that the only interaction of the user with the android device is through the user interface, where he can manage all the future operations that his device will deal with.

- User – Any android device owner that wishes to save himself the burdens of setting his device manually, over and over again.
- SmardDroid App – The core of the system, includes the UI, the processing component and the Database support.
- User Interface – A simple presentation of all the interaction the user can do in the application in terms of managing his rules.

- Database – This is an internal storage of the android device, which holds all the data and objects that the application needs in order to maintain the desired state of the device.
- Android – This refer to the device itself and all the capabilities of the system, such as, sensors for location and orientation, cellular data, and internal data of the user.
- Environment - This role represents the dynamic input data that the android device receives and translates to triggers for the android app to response accordingly.

3.2 Goals

- Develop an Android application that exposes to the user an interface to set automatic operations (Actions) in case of pre-defined events (Triggers).
- Create a smooth and simple UI and UX in order to create and maintain rules.
- Support as many triggers and actions as possible.
- Create a dynamic UI that exposing to the user, only the triggers and actions that relevant to his device, in terms of the device abilities.

3.3 SRS Document Summary

3.3.1 Overall Description

SmartDroid is a new application that replaces the operation of setting the device manually to suit the user needs for different scenarios. The application is expected to evolve over the development process, ultimately permitting the user to create smart rules that will automatically switch their devices state according to his needs and pre-requests.

3.3.2 System Features

The main feature that the SRS document concentrates on is that the user will able to:

- Create Rule.
- Edit Rule.
- Delete Rule.
- Enable/Disable Rule.
- Adding Triggers to new/existing Rule.
- Adding Actions to new/existing Rule.
- Enable/Disable specific Trigger/Action.
- Enable/Disable the entire SmartDroid application.

According to the SRS Document:

Rule reference as ‘Profile’.

Trigger reference as ‘Condition’.

3.3.3 External Interface requirements

User Interface for creating/editing/deleting/enabling/disabling Rules, Triggers and Actions.

3.3.4 Other Nonfunctional Requirements

- Battery consumption shall be lower than 10% of total usage consumption.
- Maximum support on a variety of android devices.
- The location condition awareness will have accuracy of no more than 50m radius deviation if all the location options (e.g. GPS, WIFI, GSM etc.) are available (not set off by the user).

4. Market Survey

As part of our market survey, we reviewed lots of application, frameworks and services in our field of work. We decided to concentrate on couple of applications/framework and provide the review summary:

Tasker - An android application that automates the device settings.

Pros:

- The application covers a lot of the device's functionality.
- Have a good reviews and market share.

Cons:

- The user needs to spend lots of time learning the user interface in order to become productive.
- It takes a lot of interaction with the app to set a simple rule.
- Cost money.

On{X} - Framework by Microsoft that lets you control and extend the capabilities of your Android phone using a JavaScript API to remotely program it.

Pros:

- *Nice and clean User Interface.*

Cons:

- User need have JavaScript programming knowledge to write is own rules.
- The app depends on external remote service in order of it to work.
- The code running the automation is not the device native code so it's doesn't really have the device specific capabilities.

5. Design

5.1 System Architecture

5.1.1 Architectural Description

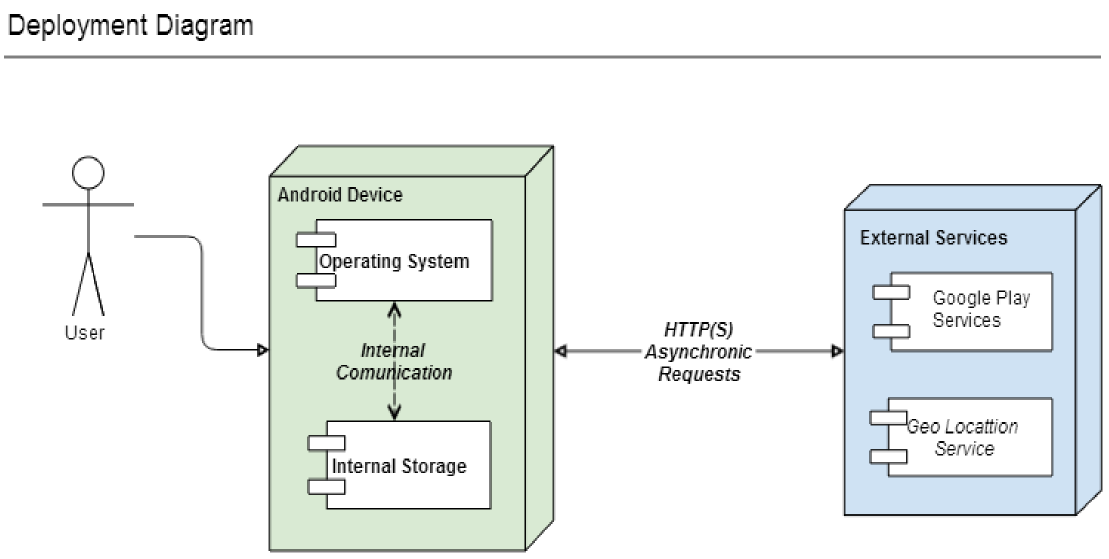


Figure 2 - Deployment Diagram.

Operating System – The android operating system runs the SmartDroid application and which preforms HTTP(S) A synchronic requests to Google Play Services and Geo Location Service.

Google Play Services – The Google Play Services contains set of convenience services for development ease like “User Activity Recognition”, a service which recognize the user activity (Driving, Cycling, Walking and Still). This service uses device hardware resources to detect the user activity, resources like cellular data, WIFI data and sensors like gyroscope, accelerometer. The Google Play Services component is installed as APK – Application Package in every android operating system since API 8.

Geo Location Service – The Geo Location Service is internal component of the android operating system providing convenience classes to perform geocoding (converting human readable address to latitude longitude) and reverse geocoding (converting latitude longitude to human readable address). This class needs the WIFI support to communicate through HTTP with Google servers for the geocoding and reverse geocoding operation, there for WIFI enabled is needed.

SmartDroid – The application component lets the user perform CRUD (Create, Read, Update and Delete) operations on rules containing triggers and actions. When the user is done manipulating the rules, the application stores the rules in OODBMS internal storage or the external storage (sdcard).

The application uses Geo Location Services to perform geocoding when the user needs to input location by translating addresses to latitude longitude.

Some of the triggers use Google Play Services to detect the user activity and determine if trigger is satisfied or not and by that decides if the rule is satisfied.

The application component is registered statically to the operating system events. The application component opens communication tunnel between itself and the operating system. Every system event the application registered to is processed by the application and if there is any stored rules that become satisfied by the system event, the rules will be perform by the application.

Internal Storage - The internal storage stores the OODBMS. The application manipulates the database by CRUD operations on DTOs like Rules, Triggers and Actions to and from the database.

5.1.2 Component Decomposition Description (package diagram)

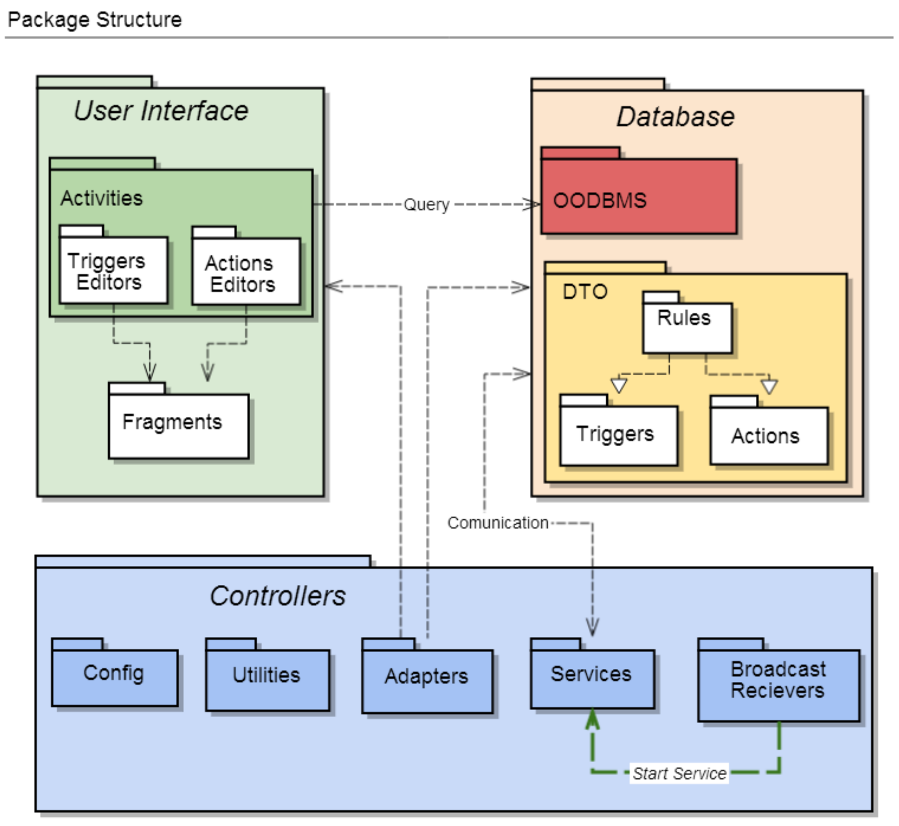


Figure 3 - Package Diagram.

The application packages designed in an MVC structure – Model, View and Controller to support Modular design with a SoC (Separation of Concerns).

The Model packages contain the Data system design structure and implementation (Data concerns).

The View packages contain the GUI system design structure and implementation (Representation UI, UX concerns).

The Controller packages control's the traffic and communication between the View and the Model (The glue that connects the Representation and Data).

The MVC structure breaks the dependencies between View and Model, thus the View or the Model can change without interfering each other.

Model Packages:

Model – Contains the Database utilities and business logic beans DTOs – Data Transfer Objects.

Database – Contains the database utilities and helper classes to communication with the OODBMS – Object Oriented Database Management System.

DTOs – Contains all the data transfer objects - Rules, Triggers and Actions.

Rules – Contains the rules data transfer object.

Actions – Contains the actions data transfer objects, the abstract class Action (Base template for future actions implementations) and the concrete actions implementations classes.

Triggers – Contains the triggers data transfer objects, the abstract class Trigger (Base template for future triggers implementations) and the concrete triggers implementations classes.

View Packages:

Activities – Contains rule activities screens the user interacts with to preform CRUD operations on rules. Rules creation workflow activities trigger editors and action editors.

TriggerEditors – Contains all the triggers editors activities screens the user interact with to preform CRUD operations on Triggers.

ActionEditors – Contains all the actions editors activities screens the user interact with to preform CRUD operations on Actions.

Fragments – Contains all the rules creation and manipulation workflow. Fragments are invoked from some of the activities.

Controller Packages:

Config – Contains the configuration class that holds the application configuration in categories and structured by inner classes with constants and convenient static methods.

Utilities – Contains the utilities classes that implements utilities static methods. There are utilities for Activity Recognition, Location and Google Play Services communication.

Adapters – Contains the adapters for list view representing rules, triggers and actions. The adapter handles data storage and manipulation for lists on screen.

Broadcast Receivers – Contains the broadcast receiver that receives the system events. The operating system instantiates broadcast receiver for every system event the application is registered to. The receiver instantiates a temporary service to process the event and passes its data to the service.

Services - Contains the service which handles registered system events. Every time a registered system event received by the application, the service processes the data and delegates the handling of the event to the right trigger.

5.1.3 Architectural Alternatives

RDBMS – Relational Database Management System

RDBMS is a database management system based on the relational model as introduced by E. F. Codd worked for IBM. Many popular databases currently in used are based on the relational database model (IBM DB2, Oracle Database, MySQL, SAP, etc.). The data in RDBMS stored in schemas and tables. Relational databases have been challenged by object databases, which were introduced in attempt to address the object-relational impedance mismatch in relational database.

OODBMS – Object Oriented Database Management System

OODMBS is a database management system combines database capabilities with object-oriented programming language capabilities. Object databases allow the object oriented developer to develop the product, store the data as objects, and replace or modify existing objects to make new objects within the OODBMS. The information in object databases is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented.

RDBMS verses OODBMS comparison: taken from http://www.nomad-ret.net/docs/rdbmsvodms-thelin.htm

Criteria	**RDBMS**	**OODBMS**
Defining standard	SQL2	ODMG-V2.0
Support for object-oriented programming	Poor; programmers spend 25% of coding time mapping the program object to the database.	Direct and extensive
Simplicity of use	Table structure easy to understand; many end-user tools available.	Ok for programmers; some SQL access for end users.
Simplicity of develop	Provides independence of data from application, good for simple relationships.	Objects are a natural way to model; Can accommodate a wide variety of types and relationships.
Extensibility and content	None	Can handle arbitrary complexity; users can write methods and on any structure.
Complex data relationships	Difficult to model	Can handle arbitrary complexity; users can write methods and on ay structure.
Performance versus interoperability	Level of safety varies with vendor; must be traded off; achieving both requires extensive testing.	Level of safety varies with vendor; most OODBMSs allow programmers to extend DBMS functionality by defining new classes.
Distribution, replication and federated databases	Extensive	Varies with vendor; a few provide extensive support.
Product maturity	Very mature	Relatively mature
Legacy people and the universality of SQL	Extensive supply of tools and trained developers	SQL accommodated, but intended for object-oriented programmers
Software ecosystems	Provided by major RDBMS companies	OODBMS vendors beginning to emulate RDBMS vendors, but none offers large markets to ISVs

Broadcast Receivers

Registration to Android operating system events listening is implemented through sub classing (deriving) broadcast receiver's class with your own implementation, and then registering your broadcast receivers to the android operating system.

There are two ways to register broadcast receivers. Static Registration verses Dynamic Registration.

Static Registration – Registering broadcast receivers to the Android operating system statically by declaring your custom broadcast receivers in the AndroidManifest.xml file with the <receiver> element tag and the intent filter actions that the broadcast is listening to. From Android 3.1 the application must be started before the Android operating system can broadcast to the application, but the broadcast will be registered to the Android operating system automatically in the APK – Android Package installation process.

Static Registration Pros:

- Simple set and forget one time registration in the AndroidManifest.xml file.
- Broadcast receivers receive events even when the application is not running. The android operating system will start the context of the application and broadcast the event data to the broadcast receiver.

Static Registration Cons:

- There are Android operating system broadcasts that cannot be registered statically but only dynamically.
- Broadcasts always received by the application from the Android operating system.
  The broadcast are not temporarily.

Dynamic Registration - Registering broadcast receivers to the Android operating system dynamically by invoking the register broadcast method on a context object done programmatically and not automatically when the APK – Android Package is installed. That said, after registering broadcast receiver programmatically the broadcast receiver must be unregistered programmatically when the application is not running at the foreground or the background, this behavior gives the dynamic registration a temporarily registration. None temporarily dynamic broadcast receiver's registration can be achieved by running a service in the background all the time, but that consumes too many valuable resources that are not spare in mobile devices.

Dynamic Registration Pros:

- Any Android operating system events can be registered by dynamic registration, but not statically registration.
- The Android operating system broadcasts can only be registered temporarily

Dynamic Registration Cons:

- The broadcast registration can only be temporarily.
- The Android operating system events listening possible only when the application is running or a long processing service is running.
- Hard to implement and maintain the state of the broadcasts registration, prone to errors and bugs when there are a lot of Android system events to listen to.

Services

Services in the Android operating system are an application component that can perform long-running operations in the background and does not provide a user interface.

Services in the Android operating system come in two flavors Service and IntentServe super classes need to be extended to custom services.

Service:

This is the base class for all services. When you extend this class, it's important that you create a new thread in which to do all the service's work, because the service uses your application's main thread, by default, which could slow the performance of any activity your application is running.

Service pros:

- Can be kept alive if needed.
- Can run tasks in parallel with threading.

Service cons:

- The service runs on the background but it runs on the main thread of the application.
- The service may block the main thread of the application.

IntentService:

This is a subclass of Service that uses a worker thread to handle all start requests, one at a time. This is the best option if you don't require that your service handle multiple requests simultaneously. Need an implementation specific method which receives the intent for each start request so you can do the background work.

IntentService Pros:

- The IntentService runs on a separate worker thread, not on the main application thread.

IntentService Cons:

- The IntentService cannot run tasks in parallel. Hence all the consecutive intents will go into the message queue for the worker thread and will execute sequentially.

5.1.4 Design Rational

SmartDroid's Architectural design consists of OODBMS, Broadcasts static registrations and IntentServie.

OODBMS:

The OODBMS object-oriented characteristics are essential for SmartDroid's object-oriented data architecture and design. SmartDroid's data design and architecture consists of complex objects relationships like aggregation, association and most of all inheritance that is the corner stones of the object-oriented programming paradigm.

The OODBMS supply naturally the object-oriented characteristics whereas RDBMS not because they are table-oriented. Choosing OODBMS in the mobile application development environment saves the object to relational mapping needed to store object to relational databases, saves the hard work in the maintenance phase of the mobile application and most of all enables the mobile application data to be as scalable as the data design and structure itself (table-oriented data design not enable this).

Broadcasts static registrations and IntentServices:

SmartDroid is a mobile application as such its development supports efficient energy consumption especially battery consumption. SmartDroid cannot afford long running services in the background, thus broadcasts dynamic registration disqualified because broadcasts dynamic registration by nature are temporarily, the work around to preserve the broadcasts dynamic registration is to run a service in the background for all the registration period (All the time). This background service consumes resources such as CPU time, RAM and of course battery which is a precious resource in mobile devices. Our efficient solution to this problem is to provide permanent broadcasts static registration and whenever an Android operation event broadcast received by the application the application will start destined short running service (IntentServie) to process the broadcast data (System event data) and to handle and operation according to the rules defined by the user in the application database, When the process of the event is finished the service will die. This development decision provides SmartDroid its USP – Unique Selling Points as the only energy efficient device automation application in the Google Play Store, because all the other application use a background service which runs all the time which drains the device battery very quickly.

5.2 Data Design

5.2.1 Database Description

The database is OODBMS – Object Oriented Database Management System (No SQL) because The RDBMS – Relational Database Management System (SQL) paradigm not supporting object oriented characteristics like inheritance and polymorphism which needed for the development of SmartDroid.

SmartDroid developed in mind to implement simple solid and concise Infrastructure for adding concrete triggers and actions as needed with ease and without changing the core infrastructure structure and code (maximum scalability).

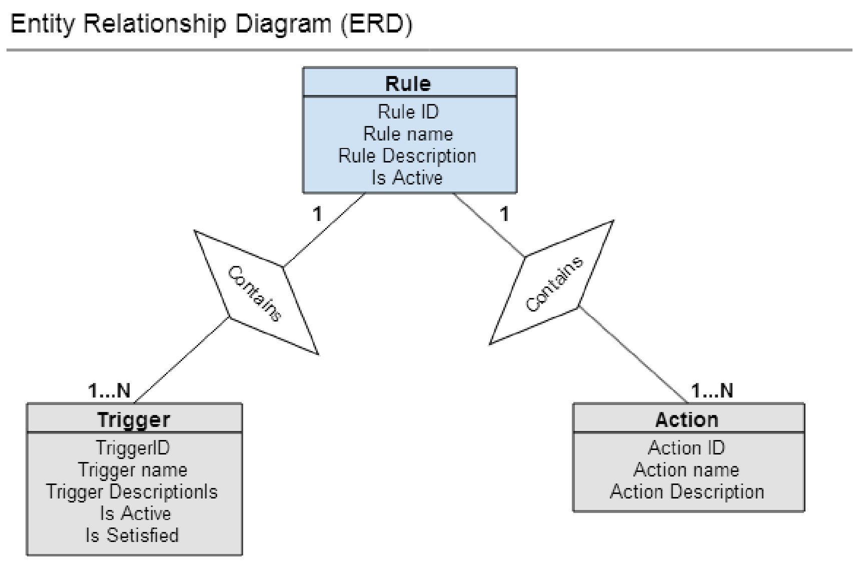


Figure 4 - ODBMS Entity Relationship Diagram (ERD)

5.2.2 Global Data Structure

List interface is the major part of the system. The data building blocks of the application rules, triggers and actions are organized in lists. The user maintains a list of rules. Every rule contains a list of triggers and a list of actions. The application entities stored in the OODBMS in Lists data structures. Trigger and actions are past between the application components in lists.

The Triggers and actions the application offers to the user when creating new rule by adding it triggers and actions are organized in lists.

5.2.3 Data Dictionary

Class name: Rule	
Brief description: The Rule class is responsible for aggregating lists of triggers and actions. The rule is a container for triggers and actions. When all the triggers satisfied meaning the rule is satisfied then then the rule is performed meaning all the actions are performed.	
Attributes (fields)	**Attribute Description**
UUID id	Represents the rule’s universal unique id.
String name	Represents the rule’s name.
String description	Represents the rule’s description.

Boolean active	Represents the rule’s activation state, if the rule is active true, otherwise false.
List<Trigger> triggers	Represents the rule’s triggers as a list.
List<Action> actions	Represents the rule’s actions as a list.
Methods (operations)	**Method Description**
Rule(String name, String description)	The rule’s class constructor.
getId()	Returns the rule’s id.
getName()	Returns the rule’s name.
setName(String name)	Sets the rule’s name.
getDescription()	Returns the rule’s description.
setDescription(String description)	Sets the rule’s description.
isActive()	Returns the rule’s activation state. If the rule is active then returns true, otherwise false.
setActive(Boolean active)	Sets the rule’s activation state. If the rule is active then true, otherwise false.
getTriggers()	Returns the rule’s triggers as a list of triggers.
setTriggers(List<Trigger> triggers)	Sets the rule’s triggers as a list of triggers.
getActions()	Returns the rule’s actions as a list of actions.
setActions(List<Action> actions)	Sets the rule’s actions as a list of actions.
isSatisfied()	Returns if the rule is satisfied meaning all its triggers are satisfied.
Perform()	Performs every action in the rule’s actions list.
Equals(Object another)	Returns true if lhs rule is equal to rhs rule by ids, otherwise false.
compareTo(Rule another)	Compares one rule to another by ids and return integer higher than 0 if lhs before rhs, 0 if the ids are equal and lower than 0 if lhs after rhs.

Class name: Action (Abstract)	
Brief description: The Action class is responsible for abstraction of the concrete actions.	
Attributes (fields)	**Attribute Description**
UUID id	Represents the action’s universal unique id.
String name	Represents the action’s name.
String description	Represents the action’s description.
Methods (operations)	**Method Description**

Action(String name, String description)	The action's class constructor.
getId()	Returns the action's id.
getName()	Returns the action's name.
setName(String name)	Sets the action's name.
getDescription()	Returns the action's description.
setDescription(String description)	Sets the action's description.
Perform()	Performs the action. Every derived class should implement this method. This is where the action logic implemented.
getIconId()	Returns the icon id represents the action.
getActionEditor()	Returns the action editor class.
Equals(Object another)	Returns true if lhs action is equal to rhs action by ids, otherwise false.

Class name: ChangeWIFIStateAction (example of the concrete action classes)	
Brief description: The ChangeWIFIStateAction class is responsible turning on and off the WIFI of the device.	
Attributes (fields)	**Attribute Description**
UUID id	Represents the action's universal unique id.
String name	Represents the action's name.
String description	Represents the action's description.
Int wantedWIFIState	Represents the wanted WIFI state on or off.
Methods (operations)	**Method Description**
ChangeWIFIStateAction()	The change WIFI state action's default constructor.
ChangeWIFIStateAction(String name, String description)	The change WIFI state action's class constructor.
getId()	Returns the action's id.
getName()	Returns the action's name.
setName(String name)	Sets the action's name.
getDescription()	Returns the action's description.
setDescription(String description)	Sets the action's description.
getWantedWIFIState(0	Returns the wanted WIFI state enabled or disabled constants represents WIFI on or off.
setWantedWIFIState(int wantedWIFIState)	Sets the wanted WIFI state enabled or disabled by a given constants representing if the WIFI should be turned on or off.
Perform()	Performs the action. Every derived class should implement this method. This is

	where the action logic implemented.
getIconId()	Returns the icon id represents the action.
getActionEditor()	Returns the action editor class.
Equals(Object another)	Returns true if lhs action is equal to rhs action by ids, otherwise false.

Class name: Trigger	
Brief description: The Trigger class is responsible for abstraction of the concrete triggers.	
Attributes (fields)	**Attribute Description**
UUID id	Represents the trigger’s universal unique id.
String name	Represents the trigger’s name.
String description	Represents the trigger’s description.
Boolean active	Represents the trigger’s activation state, if the trigger is active true, otherwise false.
Boolean satisfied	Represents the trigger’s satisfaction state, if the trigger is satisfied true, otherwise false.
Methods (operations)	**Method Description**
Trigger(String name, String description)	The trigger’s class constructor.
getId()	Returns the trigger’s id.
getName()	Returns the trigger’s name.
setName(String name)	Sets the trigger’s name.
getDescription()	Returns the trigger’s description.
setDescription(String description)	Sets the trigger’s description.
isActive()	Returns the trigger’s activation state. If the trigger is active then returns true, otherwise false.
setActive(Boolean active)	Sets the trigger’s activation state. If the trigger is active then true, otherwise false.
isSatisfied()	Returns if the rule is satisfied meaning all its triggers are satisfied.
setSatisfied(Boolean satisfied)	Sets the trigger’s satisfaction state. If the trigger is satisfied then true, otherwise false.
Register()	Registers the trigger to the operating system.
Unregister()	Unregisters the trigger from the operating system.
Handle()	Handles the trigger. Get the event from

	the operating system and decides if the trigger is satisfied or not updating his satisfaction state and preforming satisfied rules.
getIconId()	Returns the icon id represents the trigger.
getTriggerEditor()	Returns the trigger editor class.
Equals(Object another)	Returns true if lhs trigger is equal to rhs trigger by ids, otherwise false.

Class name: BatteryLowTrigger (example of the concrete trigger classes)	
Brief description: The BatteryLowTrigger class handles the battery low operating system event when android battery percentage gets under 14%.	
Attributes (fields)	**Attribute Description**
UUID id	Represents the trigger's universal unique id.
String name	Represents the trigger's name.
String description	Represents the trigger's description.
Boolean active	Represents the trigger's activation state, if the trigger is active true, otherwise false.
Boolean satisfied	Represents the trigger's satisfaction state, if the trigger is satisfied true, otherwise false.
Methods (operations)	**Method Description**
BatteryLowTrigger()	The battery low trigger's class default constructor.
BatteryLowTrigger(String name, String description)	The battery low trigger's class constructor.
getId()	Returns the trigger's id.
getName()	Returns the trigger's name.
setName(String name)	Sets the trigger's name.
getDescription()	Returns the trigger's description.
setDescription(String description)	Sets the trigger's description.
isActive()	Returns the trigger's activation state. If the trigger is active then returns true, otherwise false.
setActive(Boolean active)	Sets the trigger's activation state. If the trigger is active then true, otherwise false.
isSatisfied()	Returns if the rule is satisfied meaning all its triggers are satisfied.
setSatisfied(Boolean satisfied)	Sets the trigger's satisfaction state. If the trigger is satisfied then true, otherwise false.

Register()	Registers the trigger to the operating system.
Unregister()	Unregisters the trigger from the operating system.
Handle()	Handles the low battery event return by the operating system android when the device battery percentage gets under 14%. The trigger's satisfaction state changes to satisfy and then all the relevant satisfied rules get performed, performing their actions.
getIconId()	Returns the icon id represents the trigger.
getTriggerEditor()	Returns the trigger editor class.
Equals(Object another)	Returns true if lhs trigger is equal to rhs trigger by ids, otherwise false.

5.3 Human Interface design

5.3.1 Overview

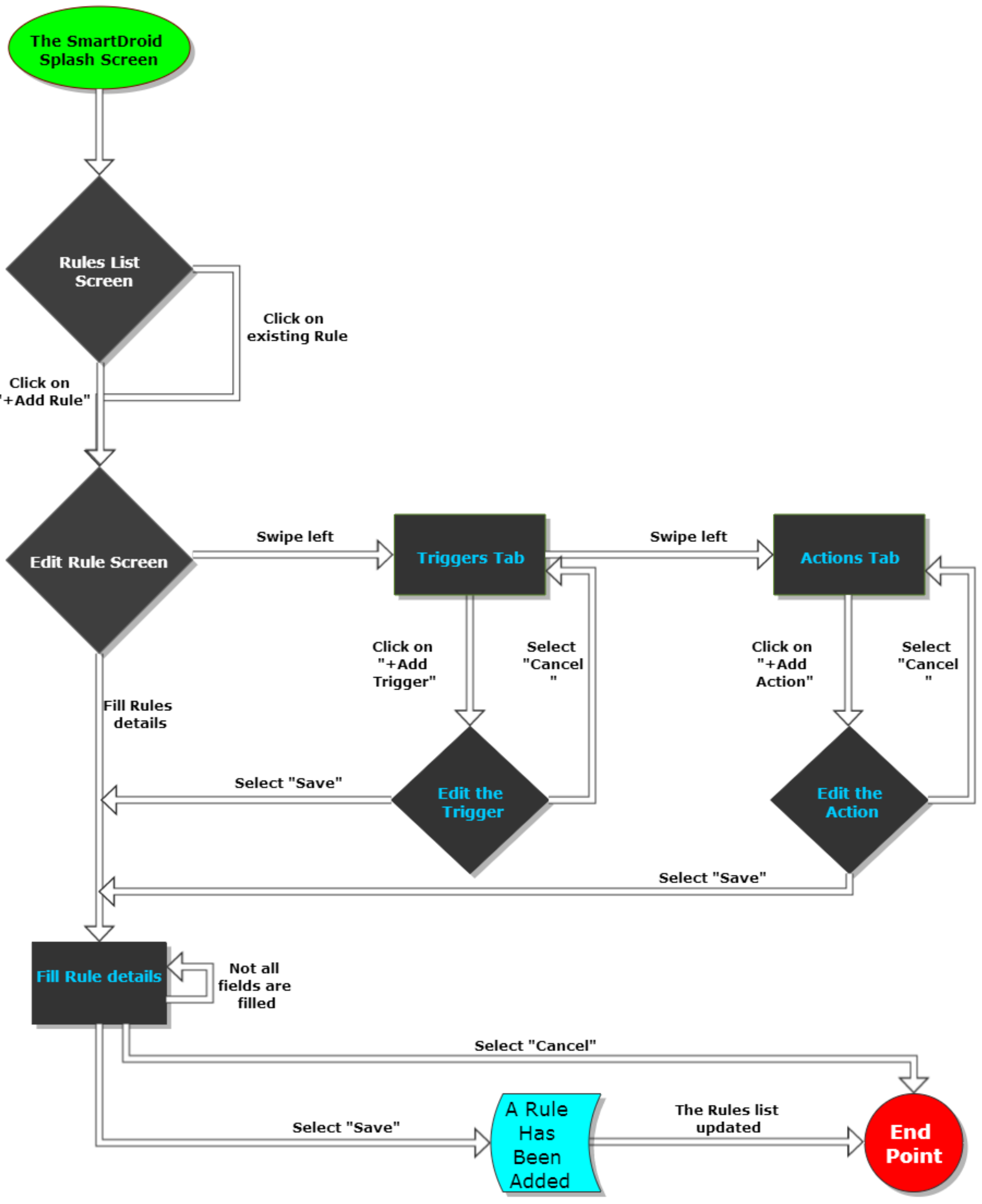
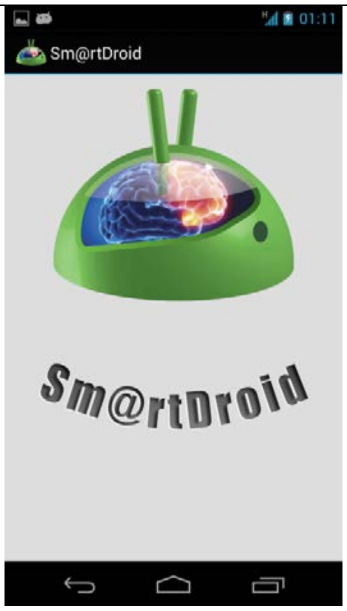
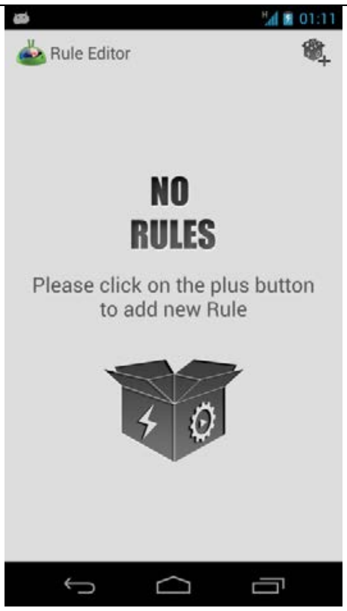
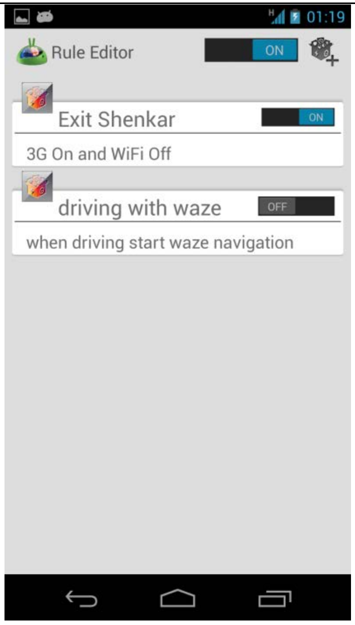


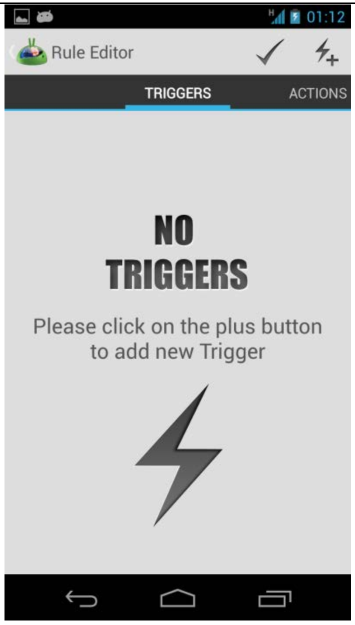
Figure 5 - User Interface flow chart for adding Rule.

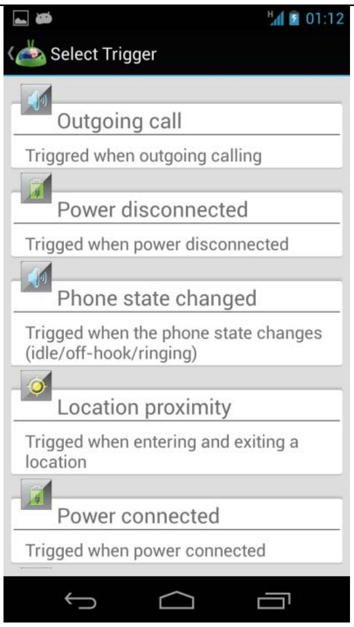
5.3.2 Screen Objects and Actions

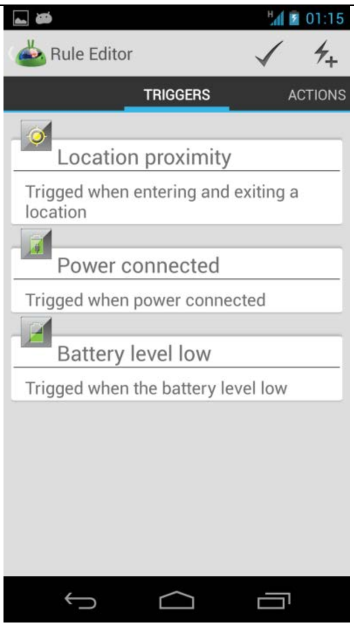
Screen Name: **Splash Screen**	
Screen Description: This is the first screen the user sees	
Way of Navigation: None	
Screen Objects and Actions	Screenshot
This page has no objects and actions It showed a 3 second animation of the logo and switches to Rule List Screen.	 Figure 6 - Splash Screen.

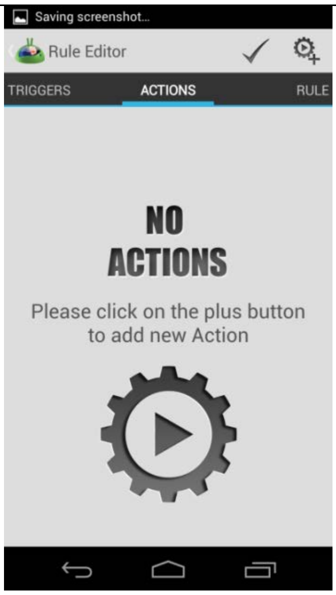
Screen Name: **Rule List Screen (without rules)**	
Screen Description: Shows list of rules the user has define (has no rules)	
Way of Navigation: Automatic navigation from the Splash Screen	
Screen Objects and Actions	Screenshot
In this screenshot the user has no rules so its display an image to indicate that state and guide the user to press the 'Add Rule' Button. **Add Rule Button**: By clicking this button the app will navigate to the 'Rule Editor' screen where the user is able to navigate through 3 screen with swipe motion left and right (Add Trigger, Add Action and Rule Information screens).	 Please click on the plus button to add new Rule Figure 7 - Empty Rules List Screen.

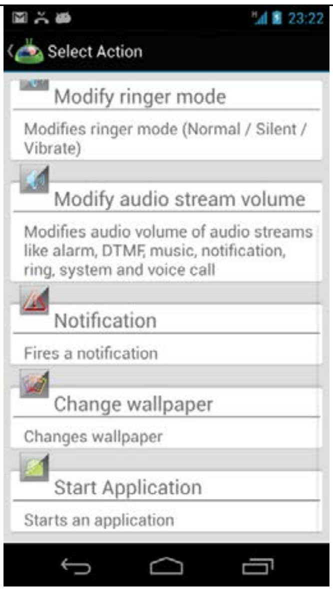
Screen Name: **Rule List Screen (with rules)**	
Screen Description: Shows list of rules the user has define	
Way of Navigation: Automatic navigation from the Splash Screen	
Screen Objects and Actions	Screen Shot
Add Rule button: By clicking this button the app will navigate to the 'Rule Editor' screen where the user is able to navigate through 3 screen with swipe motion left and right (Add Trigger, Add Action and Rule Information screens). **Rule List Items**: Clicking on rule in the rule list will navigate to the Rule Editor in Edit mode for the same rule that been clicked. ON Enable/Disable All The application rules, or Enable/Disable specific rules, by swiping the On/Off toggle button.	 01:19 Rule Editor ON Exit Shenkar ON 3G On and WiFi Off driving with waze OFF when driving start waze navigation Figure 8 - Rules List Screen.

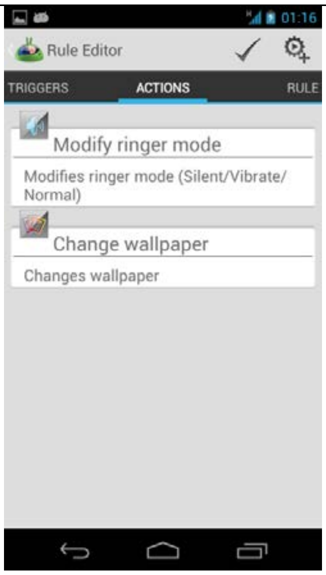
Screen Name: **Trigger List Screen (without triggers)**	
Screen Description: Shows list of triggers for specific rule (has no triggers)	
Way of Navigation: From the Rule list by clicking add new rule button.	
Screen Objects and Actions	Screen Shot
In this screenshot the user has no Triggers so its display an image that indicate that state and guide the user to press the 'Add Trigger' Button. **Add Trigger Button**: By clicking this button the app will navigate to the 'Select Trigger' screen where the can add triggers to the rule. **Save Rule Button:** By clicking this button the user can save the rule. In case that the rule is missing components, a message will pop that contains the missing component.	 01:12 Rule Editor TRIGGERS ACTIONS NO TRIGGERS Please click on the plus button to add new Trigger Figure 9 - Empty Triggers List Screen.

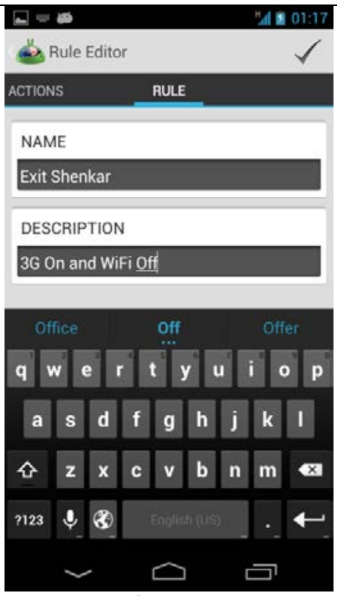
Screen Name: **Select Trigger Screen**	
Screen Description: This screen allows the user to choose triggers from a list of triggers.	
Way of Navigation: By clicking the ‘Add Trigger’ button in the ‘Trigger List’ screen.	
Screen Objects and Actions	Screen Shot
Trigger List Item: Selecting a trigger from the list by click, will navigate to the trigger type editor screen.	 01:12 Select Trigger Outgoing call Triggred when outgoing calling Power disconnected Trigged when power disconnected Phone state changed Trigged when the phone state changes (idle/off-hook/ringing) Location proximity Trigged when entering and exiting a location Power connected Trigged when power connected Figure 10 - Select Trigger Screen.

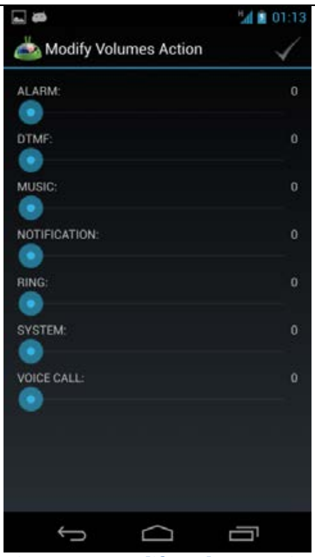
Screen Name: **Trigger List Screen (with triggers)**	
Screen Description: Shows list of triggers for specific rule.	
Way of Navigation: From the ‘Rule Editor’ screen by clicking on existing rule.	
Screen Objects and Actions	Screen Shot
Add Trigger Button: By clicking this button the app will navigate to the ‘Select Trigger’ screen where they can add triggers to the rule. **Save Rule Button:** By clicking this button the user can save the rule. In case that the rule is missing components, a message will pop that contains the missing component. **Trigger List Item**: Editing trigger from the list by click, will navigate to the trigger type editor screen.	 01:15 Rule Editor TRIGGERS ACTIONS Location proximity Trigged when entering and exiting a location Power connected Trigged when power connected Battery level low Trigged when the battery level low Figure 11 - Trigger List Screen.

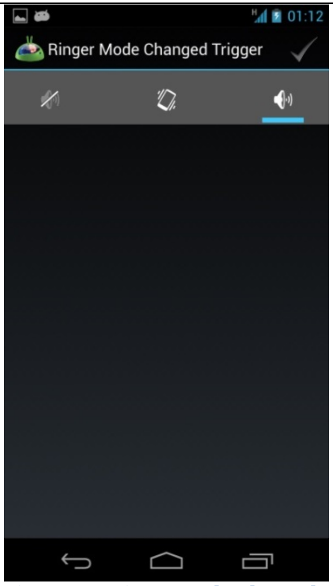
Screen Name: **Action List Screen (without actions)**	
Screen Description: Shows list of actions for specific rule (has no actions)	
Way of Navigation: From the ‘Trigger List’ screen by swipe left gesture.	
Screen Objects and Actions	Screen Shot
In this screenshot the user has no Actions so its display an image that indicate that state and guide the user to press the ‘Add Action’ Button. **Add Action Button**: By clicking this button the app will navigate to the ‘Select Action’ screen where the can add actions to the rule. **Save Rule Button:** By clicking this button the user can save the rule. In case that the rule is missing components, a message will pop that contains the missing component.	 Saving screenshot... Rule Editor TRIGGERS ACTIONS RULE NO ACTIONS Please click on the plus button to add new Action Figure 12 - Empty Actions List Screen.

Screen Name: **Select Action Screen**	
Screen Description: This screen allows the user to choose actions from a list of actions.	
Way of Navigation: By clicking the ‘Add Action button in the ‘Action List’ screen.	
Screen Objects and Actions	Screen Shot
Action List Item: Selecting an action from the list by click, will navigate to the action type editor screen.	23:22 Select Action Modify ringer mode Modifies ringer mode (Normal / Silent / Vibrate) Modify audio stream volume Modifies audio volume of audio streams like alarm, DTMF, music, notification, ring, system and voice call Notification Fires a notification Change wallpaper Changes wallpaper Start Application Starts an application Figure 13 - Select Action Screen.

Screen Name: **Action List Screen (with actions)**	
Screen Description: Shows list of actions for specific rule.	
Way of Navigation: From the 'Rule Editor' screen by clicking on existing rule and swiping left.	
Screen Objects and Actions	Screen Shot
Add Action Button: By clicking this button the app will navigate to the 'Select Action' screen where they can add triggers to the rule. **Save Rule Button:** By clicking this button the user can save the rule. In case that the rule is missing components, a message will pop that contains the missing component. **Action List Item**: Editing Action from the list by click, will navigate to the Action type editor screen.	 Rule Editor TRIGGERS ACTIONS RULE Modify ringer mode Modifies ringer mode (Silent/Vibrate/ Normal) Change wallpaper Changes wallpaper Figure 14 - Actions List Screen.

Screen Name: **Rule Details Screen**	
Screen Description: The screen where the user inserts details of the rule for identification.	
Way of Navigation: From the 'Action List' screen by swipe left gesture.	
Screen Objects and Actions	Screen Shot
Save Rule Button: By clicking this button the user can save the rule. In case that the rule is missing components, a message will pop that contains the missing component. **Name Field**: In the name field the user type with the soft keyboard the name he likes for the rule. **Description Field**: In the description field the user type with the soft keyboard the description he likes for the rule.	 Rule Editor ACTIONS RULE NAME Exit Shenkar DESCRIPTION 3G On and WiFi Off Figure 15 - Rule's Description Screen.

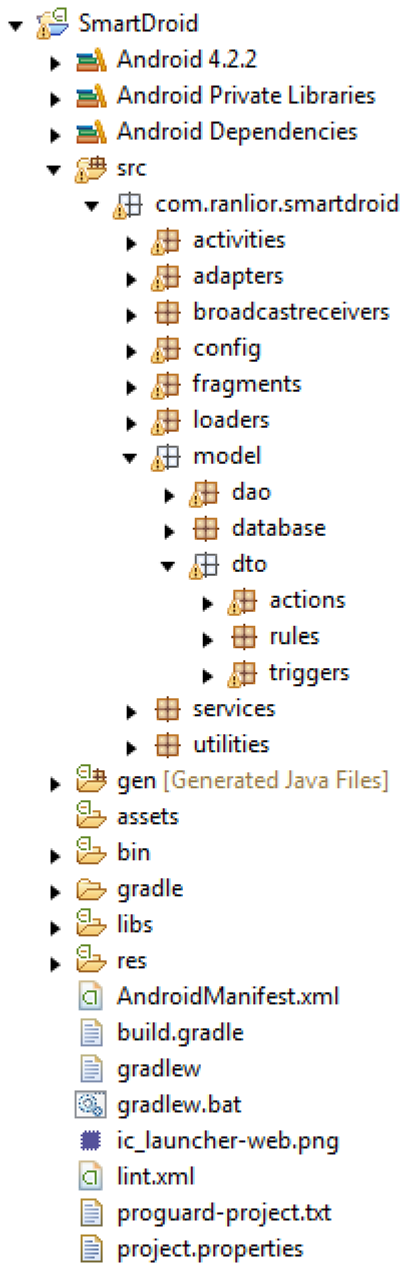
Screen Name: **Modify Volume Action Screen**	
Screen Description: This is a specific action configuration screen.	
Way of Navigation: By selecting an action from the ‘Select Action’ screen.	
Screen Objects and Actions	Screen Shot
This is just an example of configuration page that is unique for the ‘Modify Audio Stream’ Action. ✓ **Save Action Button:** By clicking this button the user can save the action. **Seek Bars Set**: the screens composed of seek bars the can set values of different audio streams by dragging the handles left or right.	 01:13 Modify Volumes Action ALARM: 0 DTMF: 0 MUSIC: 0 NOTIFICATION: 0 RING: 0 SYSTEM: 0 VOICE CALL: 0 Figure 16 - Modify Volume Action Screen.

Screen Name: **Ringer Mode Change Configuration Screen**	
Screen Description: This is a specific trigger configuration screen.	
Way of Navigation: By selecting a trigger from the ‘Select Trigger screen.	
Screen Objects and Actions	Screen Shot
This is just an example of configuration page that is unique for the ‘Ringer Mode Changed’ Trigger. ✓ **Save Trigger Button:** By clicking this button the user can save the trigger. **Iconized Radio Button Set**: the screens composed of three radio buttons that represents the different ringer mode of android system (normal, silent, vibrate).	 01:12 Ringer Mode Changed Trigger Figure 17 - Ringer Mode Changed Trigger Screen.

6. Implementation

6.1 Project Structure

The project is well formed and organized in packages. This kind of structure is the best practice when it's come to project maintenance. Figure 18, shows a snapshot of our application structure.



```
SmartDroid
    Android 4.2.2
    Android Private Libraries
    Android Dependencies
    src
        com.ranlior.smartdroid
            activities
            adapters
            broadcastreceivers
            config
            fragments
            loaders
            model
                dao
                database
                dto
                    actions
                    rules
                    triggers
            services
            utilities
    gen [Generated Java Files]
    assets
    bin
    gradle
    libs
    res
    AndroidManifest.xml
    build.gradle
    gradlew
    gradlew.bat
    ic_launcher-web.png
    lint.xml
    proguard-project.txt
    project.properties
```

Figure 18 - SmartDroid project hierarchy as it appear in "Eclipse IDE".

Under the “src” folder, it’s possible to see our classification of packages as they presented in **section 7.1.2.**

6.2 Following the Activity Lifecycle

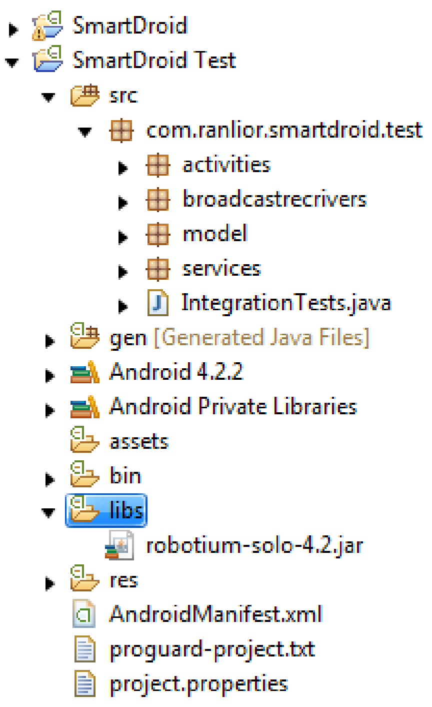
All the activities and fragments classes are implemented according to the “**Activity Lifecycle**”, which information about that can be found in ***Appendix A*** – at the end of this document.

7. Quality Assurance

As part of our quality assurance process we took couple of measures:

7.1 Automation Testing

The Automation Testing phase of this project was built as another android project that called “SmartDroid Test” which contains all the testing code and scenarios for the SmartDroid application.



```
▸ SmartDroid
▾ SmartDroid Test
    ▾ src
        ▾ com.ranlior.smartdroid.test
            ▸ activities
            ▸ broadcastrecrivers
            ▸ model
            ▸ services
            ▸ IntegrationTests.java
    ▸ gen [Generated Java Files]
    ▸ Android 4.2.2
    ▸ Android Private Libraries
      assets
    ▸ bin
    ▾ libs
        robotium-solo-4.2.jar
    ▸ res
      AndroidManifest.xml
      proguard-project.txt
      project.properties
```

Figure 19 - SmartDroid Test project hierarchy as it appear in "Eclipse IDE".

Robotium Automation Framework – In figure 19 under the libs folder in the testing project hierarchy, is the “robotium-solo-4.2.jar” file which is the java library that we used in order to develop the SmarDroid testing project.

The test project, just like the SmartDroid project, is arranged in packages for easy maintenance. Each package holds the classes that contain the testing code for the original SmartDroid project packages.

The SmartDroid Test project contains tests that cover UI, Database, Logic, and Integration test cases.

The “activities” package contains source code for UI testing.

The “broadcastrecievers” and “services” packages contain source code for Logic testing.

The “model” package contains source code for Database testing.

The “IntegrationTests” class contains an “End to end” testing and scenarios.

7.2 Manual QA

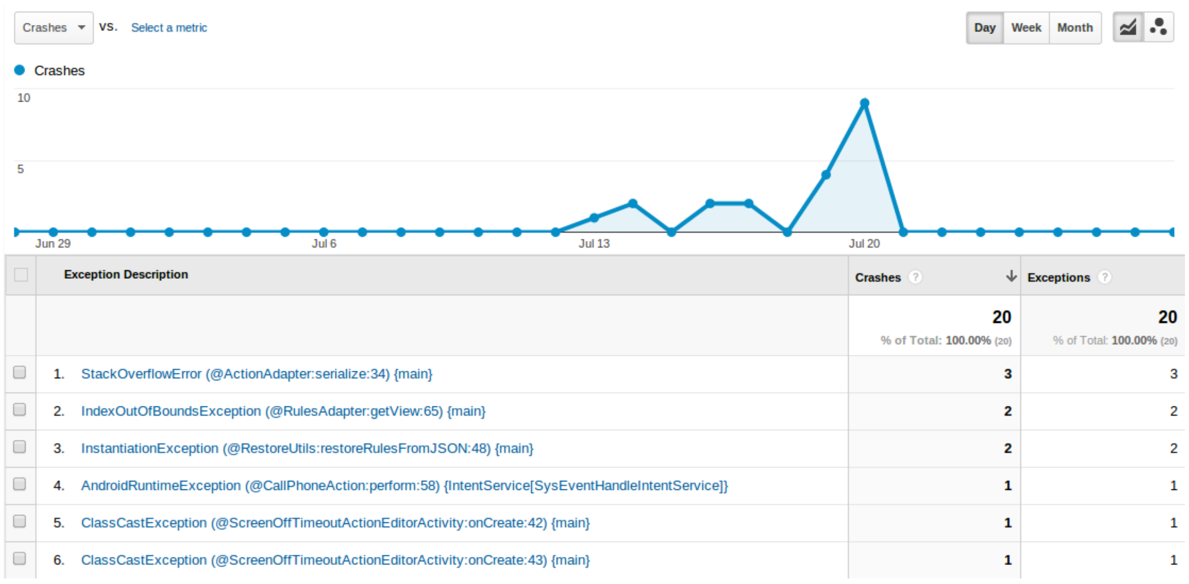
Manual QA was basically the first part of our testing phase, when the application developing process was in its earliest part, and there were not too much to test.

There was no test plan, and each test case was executed randomly, and we didn't keep result data or analyses

As the project scale, we realized that Manual QA is time consuming and test cases are been left behind, we then decided that it's time to stabilize our testing system and automation testing was established.

Manual QA testing was reserved only for test cases were automation testing could not provide us reliable result, such as: User Activity Testing, Location Based Services.

7.3 Google Analytics as Crash Reporting and Monitoring tool



	Exception Description	Crashes	Exceptions
		20 % of Total: 100.00% (20)	20 % of Total: 100.00% (20)
1.	StackOverflowError (@ActionAdapter:serialize:34) {main}	3	3
2.	IndexOutOfBoundsException (@RulesAdapter:getView:65) {main}	2	2
3.	InstantiationException (@RestoreUtils:restoreRulesFromJSON:48) {main}	2	2
4.	AndroidRuntimeException (@CallPhoneAction:perform:58) {IntentService[SysEventHandleIntentService]}	1	1
5.	ClassCastException (@ScreenOffTimeoutActionEditorActivity:onCreate:42) {main}	1	1
6.	ClassCastException (@ScreenOffTimeoutActionEditorActivity:onCreate:43) {main}	1	1

Figure 20 - Google Analytics crash report view.

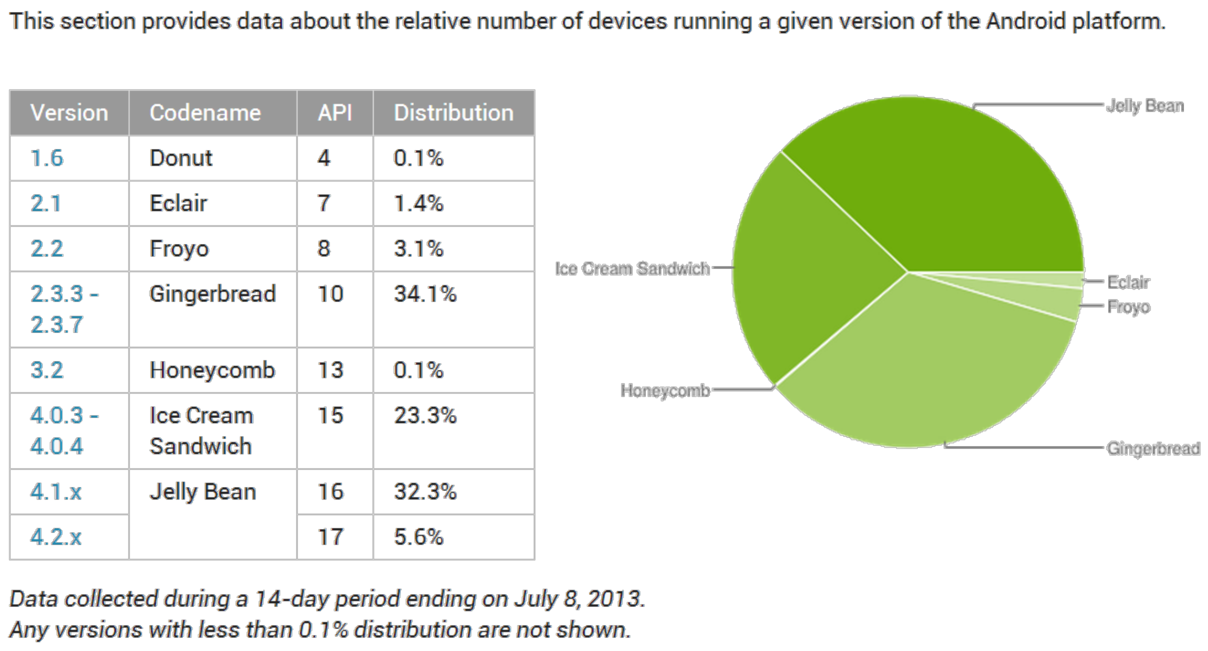
Google Analytics is an amazing tool for crash reporting and other information about our application behavior. In figure 20, we can see a graph, representing the amount of exceptions that leaded crash of our application and a table of details that include:

- Class name, function name and line number, where the exception was thrown from.
- The number of crashes / exceptions by the reference type.

Because Google Analytics works during any phase of the development process, including our nightly automation builds, a sets of negative testing led to a lot of crashes in short amount of time as can be notice in the graph section.

8. Evaluation

8.1 Android API Support


This section provides data about the relative number of devices running a given version of the Android platform.

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.4%
2.2	Froyo	8	3.1%
2.3.3 - 2.3.7	Gingerbread	10	34.1%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	23.3%
4.1.x	Jelly Bean	16	32.3%
4.2.x		17	5.6%

Data collected during a 14-day period ending on July 8, 2013.
Any versions with less than 0.1% distribution are not shown.

Figure 21 - API Version Vs. Distribution table.

SmartDroid application currently support's API Levels 7-17 which is 98.5% of all android devices distributions worldwide. This was a part of the requirements that we put in high priority and consider it a "MUST" requirement.

8.2 Battery and Memory Consumption

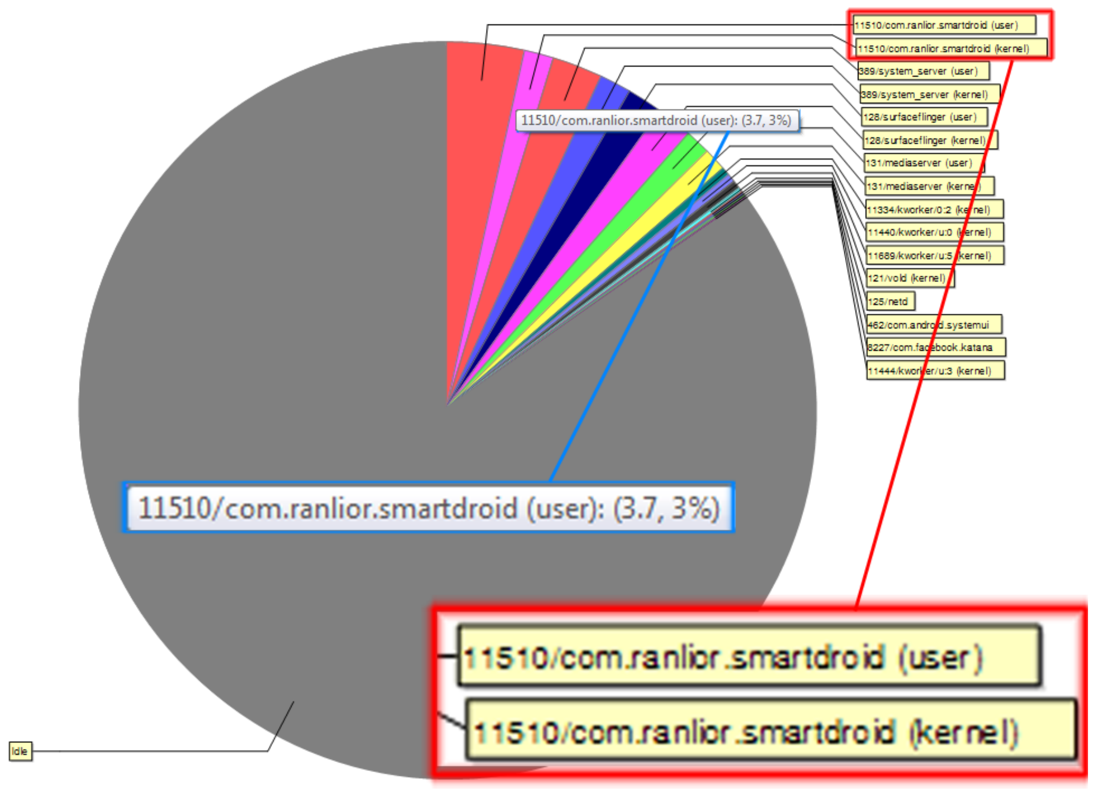


Figure 22 - Battery relative consumption.

SmartDroid low battery consumption was also a major non-functional requirement that we spent a lot of focus and care in order to achieve. The desire result of “Under 10% of the overall battery consumption of the device at given time when the SmartDroid application is running”, was over whelming as we are witnessing a snapshot of real-time battery usage of testing device of type “Samsung Galaxy Nexus”, which is running our Smartdroid application, and as we can see, the process name shown in the red rectangle, and the ***battery usage of 3.7%*** shown in the blue rectangle (***Figure 21***)

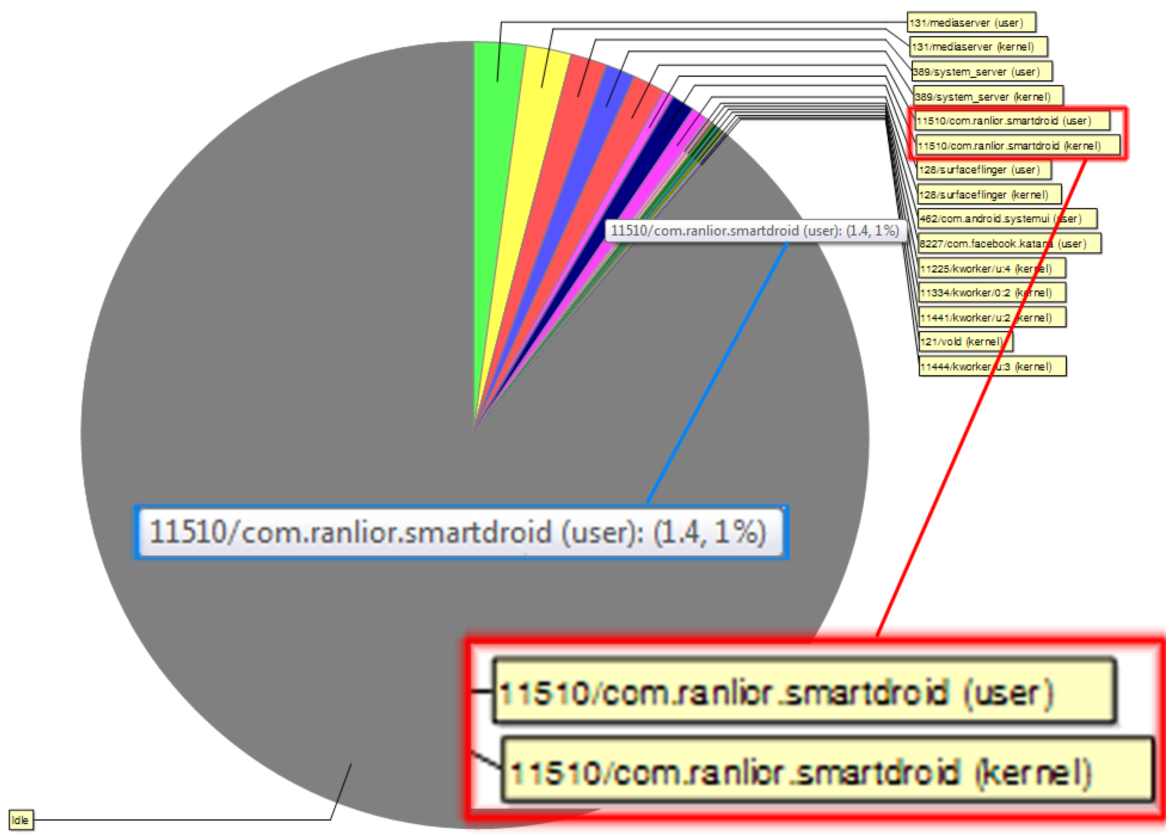


Figure 23 - Memory relative consumption.

SmartDroid memory consumption was also a major non-functional requirement that we spent a lot of focus and care in order to achieve. The desire result was not specified cause estimating the memory usage is changing from device to another and require a higher level of system kernel understanding. A snapshot of real-time memory usage of testing device of type “Samsung Galaxy Nexus”, which is running our Smartdroid application, showed relatively ***average memory consumption of 1.4%*** in compare to other system kernel process usage. (***Figure 22***)

8.3 Google Analytics as Evaluation and Summery tool

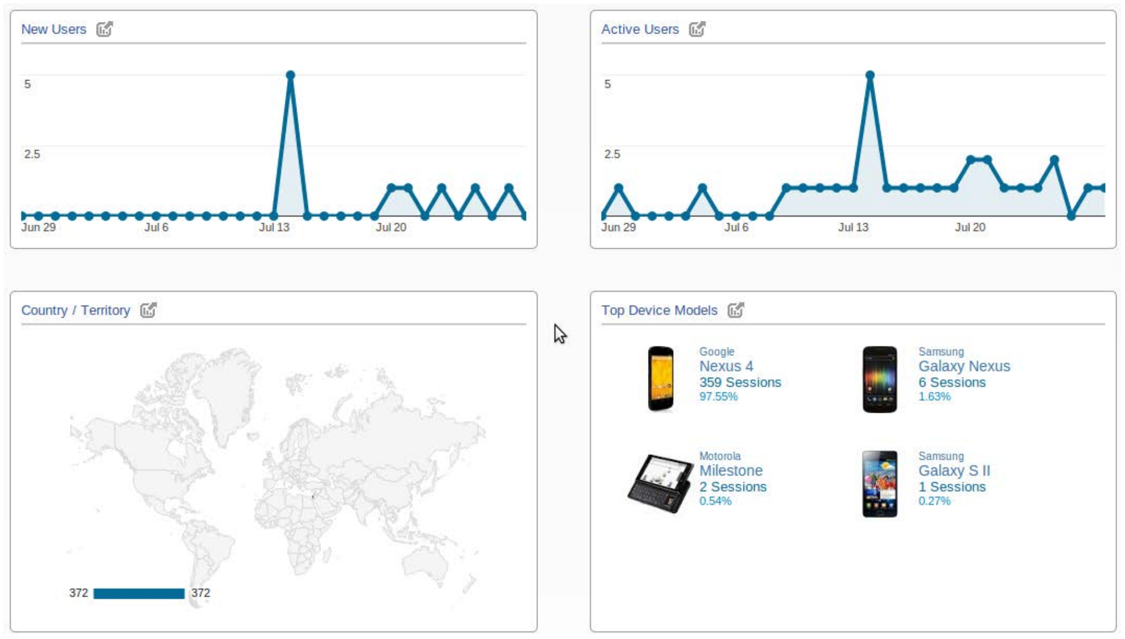


Figure 24 - Google Analytics Dashboard view.

Google Analytics provided a lot of information about our alpha version users. In figure 24 at the bottom right panel we can see the verity of devices models that have been running our application. The Google NEXUS 4 was our personal device which used to test our application.

The other panels providing information such as, New Users per day, Active Users which currently (while taking the snapshot) using our app, and because we have not distributed our application yet, so it’s possible to see that Country/Territory map panel points to Israel which is the only country right now that have users who’s install the application on their devices.

9. Future Work

- Cover as many features of the current and upcoming devices in terms of rules and triggers.
- Create a sophisticated rules sharing system over the web and between our users.
- Get involve with a professional application designer in order to improve the user experience.

10. Appendix A – The Activity Lifecycle

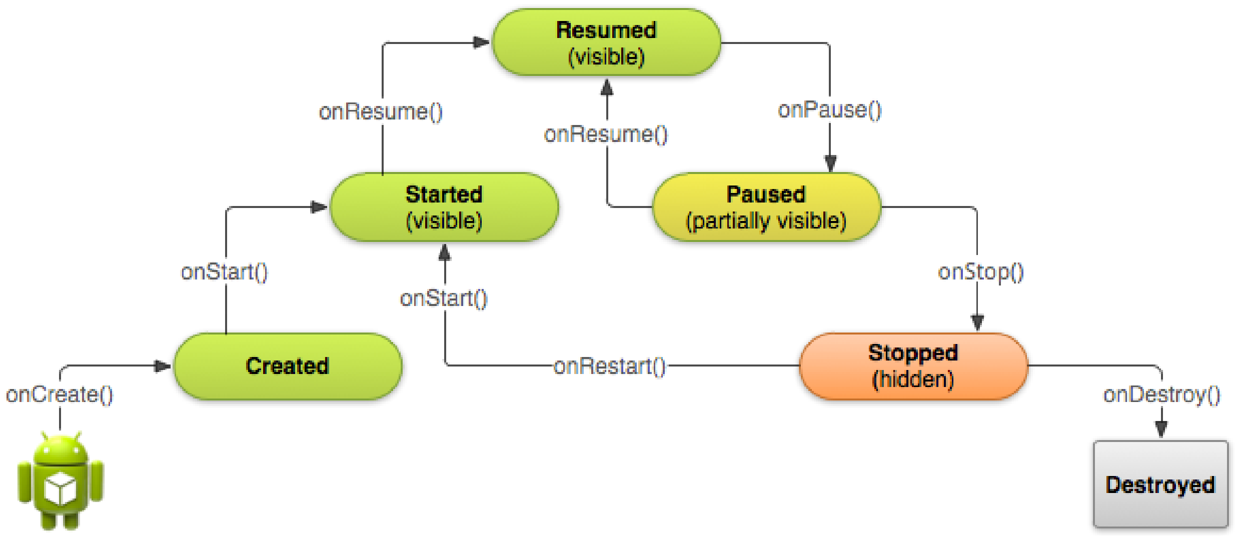


Figure 25 - A simplified illustration of the Activity lifecycle.

The Activity Lifecycle expressed as a step pyramid. This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there's a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.

Implementing all the Activity's lifecycle callback methods ensure that our app behaves the way users expect in several ways:

- Does not crash if the user receives a phone call or switches to another app while using our app.
- Does not consume valuable system resources when the user is not actively using it.
- Does not lose the user's progress if they leave your app and return to it at a later time.
- Does not crash or lose the user's progress when the screen rotates between landscape and portrait orientation.

Only three of the states illustrated in figure 23, can be static. That is, the activity can exist in one of only three states for an extended period of time:

Resumed

In this state, the activity is in the foreground and the user can interact with it. (Some time referred as the "running" state)

Paused

In this state, the activity is partially obscured by another activity. The other activity that's in the foreground is semi-transparent or doesn't cover the entire screen. The paused activity does not receive user input and cannot execute any code.

Stopped

In this state, the activity is completely hidden and not visible to the user. It is considered to be in the background. While stopped, the activity instance and all its state information such as member’s variables are retained, but it cannot execute any code.

The other states (***Created*** and ***Started***) are transient and the system quickly moves from them to the next state by calling the next lifecycle callback method. That is, after the system calls “onCreate()”, it quickly calls “onStart()”, which is quickly followed by “onResume()”.