**MAEESTRO**

1. Introduction
   1. Overview
   2. System goals
2. Proposed system
   1. Overview
   2. Current and similar System
   3. Functional requirements
   4. Non functional requirements
   5. Actors Diagram
   6. Use Cases
   7. Sequence Diagram
3. System Design
   1. System architecture (Wrapper Idea)
   2. Subsystems
   3. Layers
4. Object Design
   1. Classes and Interfaces
   2. Packages
5. **Introduction**
   1. **Overview**

One of the fastest growing business sectors is the smartphone market, of which Android has the largest share. This lead to the idea of expand the Android apps usage experience from spartphones to other systems.

* 1. **System goals**

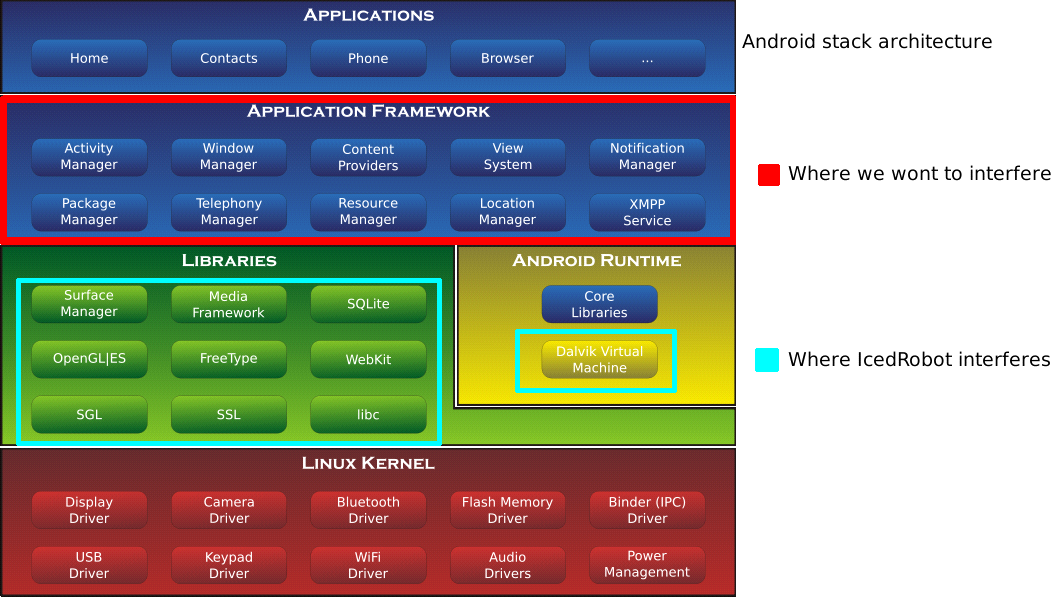
The goal of the MAEESTRO is to enable the execution of Android source code without any changes, on a JavaVM and in particular by the JamaicaVM. In this way Android apps could run on each device which has a JavaVM on any architecture.

1. **Current and similar Systems**

Actually there is any tool which meets our requirements of running Android source code.

There are few projects which goals are to run Android applications and one of these projects is IcedRobot, which substitutes the DalvikVM with a Virtual Machine called Daneel which executes and uses its own native libraries instead of the Android native libraries. Because IcedRobot uses its own Virtual Machine to run Android applications, is not reusable for our scope infact the first goal of this project is to improve the use of the JamaicaVM.

The following diagram shows the incompatible goals of IcedRobot.



1. **Proposed system**

**3.1 Overview**

MAEESTRO is a tool which provide the functionality of running and executing any code that was written for Android.

In particular, the MAEESTRO will take care of the following aspects:

* Android APIs management.
* Android graphic interface rendering.
* Device properties management.

**3.2 Functional requirements**

The following list shows the functional requirements for the MAEESTRO development:

* make the Android source code recognizable by any JavaVM,
* Execute correctly the Android event management system,
* Representation of all Android widgets by Swing objects, and
* Android Activities management.

**3.3 Non-functional requirements**

The final goal of the MAEESTRO is to run generic Android applications on embedded systems, which has the identified following non functional requirements.

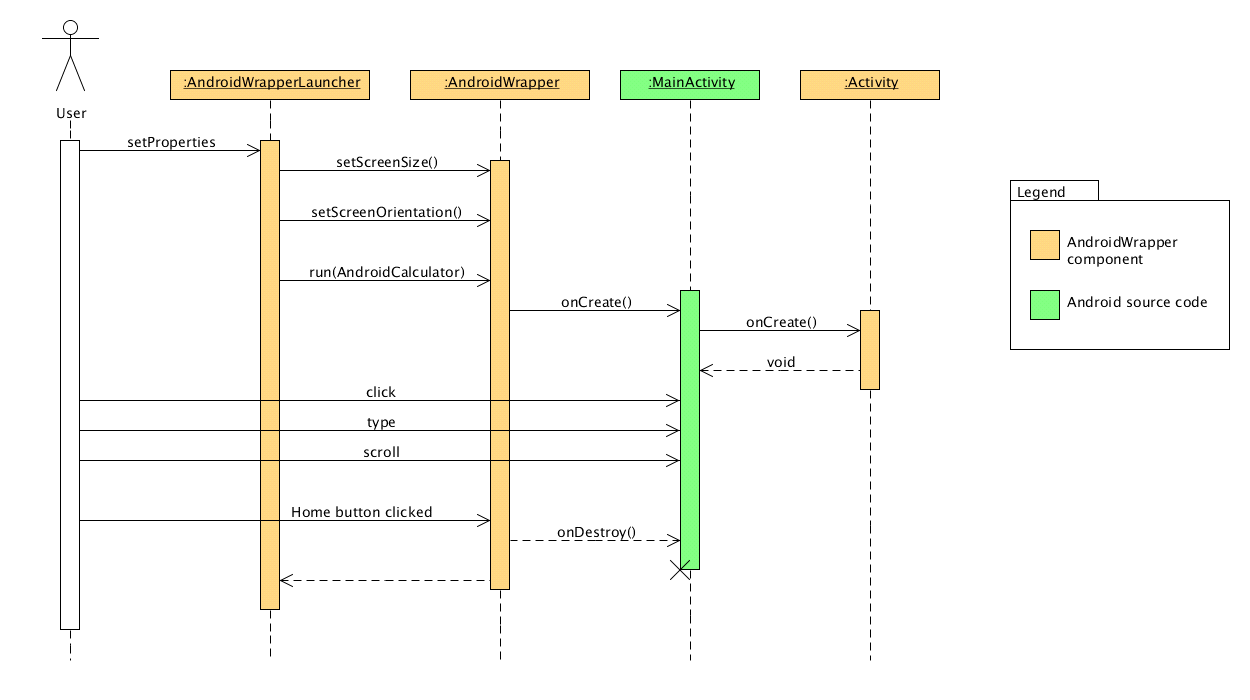
* **Performance -** the tool must ensure a good performance level for embedded systems, for this reason the implementation has to be light and needs a short reaction time.
* **Usability** -the execution of Android applications must give to the user the impression of using an Android device, so the tool has to represent the graphic interface in the same way as the original applications.
* **Multi-platform –** the Android apps have to be the same behaviour on any kind of hardware architecture, such like embedded systems or personal computer, and on the different operating systems (Windows, Linux and its derivates).

**3.5 Use Cases**

|  |  |
| --- | --- |
| **Name** | Run AndroidCalculator |
| **Entry condition** | MAEESTRO started. |
| **Event flow** | * The user insertsthe screen size and the screen orientation in the MAEESTRO Launcher window; * The user choosesthe AndroidCalculator app and clickson “Run”; * MAEESTRO executesthe AndroidCalculator application; * The user types the numbers in the EditText components and clicksthe operation button; * The AndroidCalculator returns the result; * The user clickson the Android Home button; * MAEESTRO terminatesthe AndroidCalculator and comes back to the start window; |
| **Exit condition** | Home button or Back button is pressed. |
| **Quality requirements** | The interaction between the user and the application must be the same as the interaction offered by an Android device. |

**3.6 Sequence Diagram**

The following basic sequence diagram shows an Android application executed by the MAEESTRO:



1. **System Design**

**4.1 System architecture**

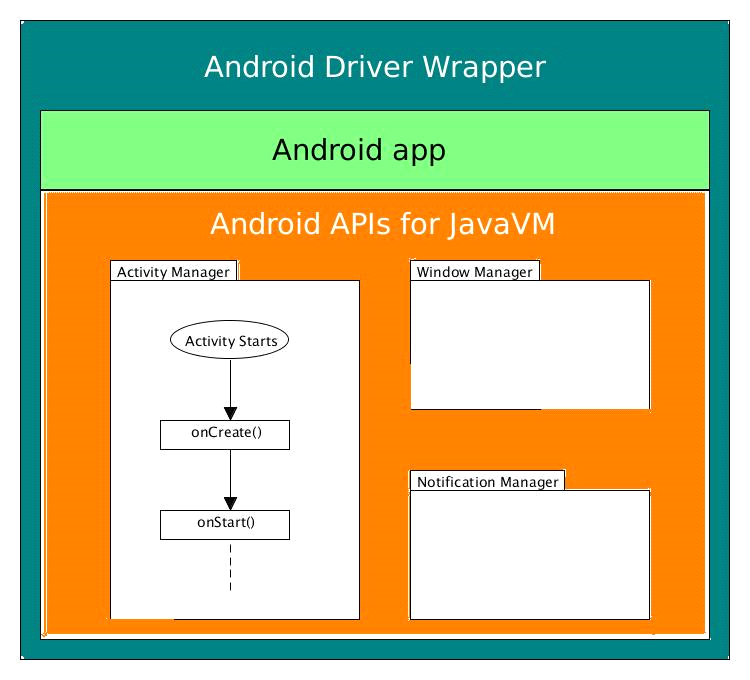
To run Android code on a different Virtual Machine, it is necessary to make the Android APIs understandable for the system. This implies that the whole Application framework of the Android stack has to be rewritten to run on the VM. The best solution for this problem is to use a Adapter Pattern.

The following scheme represents the structure of an Android application executed on a JavaVM

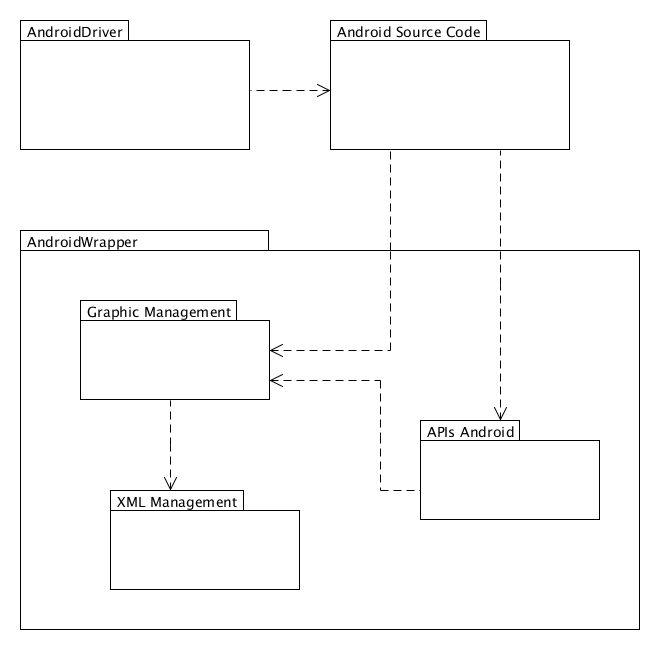


Besides, Android applications do not have the same structure as a common Java program. In fact Android applications do not have a *main* method that starts the execution but they are based on the Android Activity life-cycle, which is the entry point for Android applications. For this reason, it is needed to provide a pattern which is able to direct the execution in the right way.

The final result will be an Adapter pattern that will work as shown here:



**4.2 Subsystems**



The decomposition of the system is expected to be composed of three subsystems that manage different aspects and functions:

* Android APIs: Java implementation of the original Android APIs. It is subdivided into other subsystems following the Android operating system subdivision;
* Graphic Manager: for drawing and managing graphic components;
* XML Manager: for parsing .xml files that contains graphical details;