



**UNIVERSITY OF INDONESIA**



**UNIVERSITY OF SOUTH-  
BRITTANY**

**DEVELOPMENT OF SAJE PROJECT ON ANDROID OPERATING  
SYSTEM**

**THESIS**

**FITRI WIBOWO  
0906578314**

**FACULTY OF ENGINEERING  
DEPARTMENT OF ELECTRICAL ENGINEERING  
MULTIMEDIA AND NETWORK INFORMATION  
DEPOK  
JULI 2011**



**UNIVERSITY OF INDONESIA**



**UNIVERSITY OF SOUTH-  
BRITTANY**

**DEVELOPMENT OF SAJE PROJECT ON ANDROID OPERATING  
SYSTEM**

**THESIS**

**Submitted to the University of Indonesia  
in partial fulfillment of the requirements for  
the degree of Magister Teknik**

**FITRI WIBOWO  
0906578314**

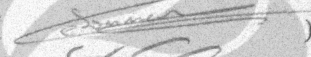



**FACULTY OF ENGINEERING  
DEPARTMENT OF ELECTRICAL ENGINEERING  
MULTIMEDIA AND NETWORK INFORMATION  
DEPOK  
JULI 2011**

## APPROVAL FORM

This Report is proposed by:

Name : Fitri Wibowo  
NPM : 0906578314  
Study Program : Electrical Engineering  
Title : Development of SAJE project in Android operating system

Has been officially approved, supervised and finally examined by the examiners board authorized by Université de Bretagne-Sud, in partial fulfillment of the requirements for the degree of Magister Teknik at Electrical Engineering Department, Technical Faculty, Universitas Indonesia.

Supervisor : Nicolas LE SOMMER (  )  
Supervisor : Patrice FRISON (  )  
Examiner : Yves MAHÉO (  )  
Examiner : Frédéric GUIDEC (  )

Approved in : Vannes, French  
Date : 4 July 2011

Acknowledged,

The Director of Formation M2 IIR,  
Université de Bretagne-Sud,  
Vannes-France



Nicolas COURTY

The Head of Electrical Engineering  
Department, Universitas Indonesia,  
Depok-Indonesia



Dr. Ir. M. Asvial, M.Eng

## DECLARATION OF ORIGINALITY

I hereby declare that this thesis and the work reported herein was composed by and originated entirely from me. Information derived from the published and unpublished work of others has been acknowledged in the text and references are given in the list of sources.

Name : Fitri Wibowo

NPM : 0906578314

Signature : 

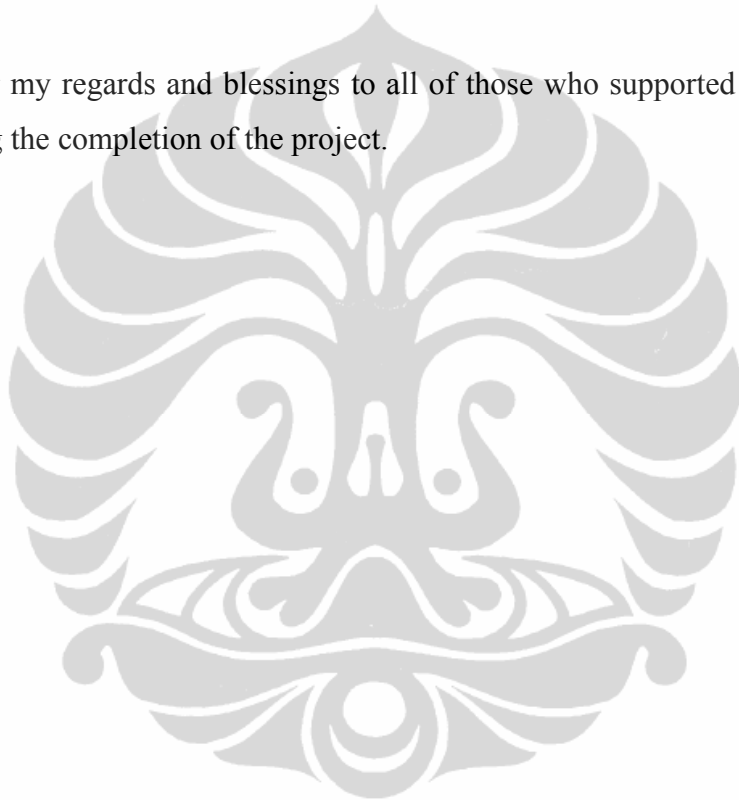
Date : 5 July 2011



## ACKNOWLEDGMENT

I would like to acknowledge the following people for their support and assistance in this thesis. From the CASA team of VALORIA, I would like to thank my supervisor , Nicolas le Sommer, for guiding me from the initial to the final level of this project. And also my tutor at University of South-Brittany, Patrice Frison, for the constant reminders and much needed motivation.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.



## DECLARATION OF PUBLICATION APPROVAL PAGE FOR ACADEMIC PURPOSE

As an civitas academic of University of Indonesia, I the undersigned below:

Name : Fitri Wibowo  
NPM : 0906578314  
Study Program : Electrical Engineering  
Department : Electrical Engineering  
Faculty : Engineering  
Type of Work : Thesis

for the development of science, agreed to give University of Indonesia a **non-exclusive royalty-free right** on my scientific work entitled:

### **Development of SAJE Project on Android Operating System**

with the existing devices (if needed). With this non-exclusive, royalty-free right, University of Indonesia has permission to copy, re-distribute in different format and can publish this Thesis by including my name as a writer and copyright owner.

Vannes, 5 July 2011

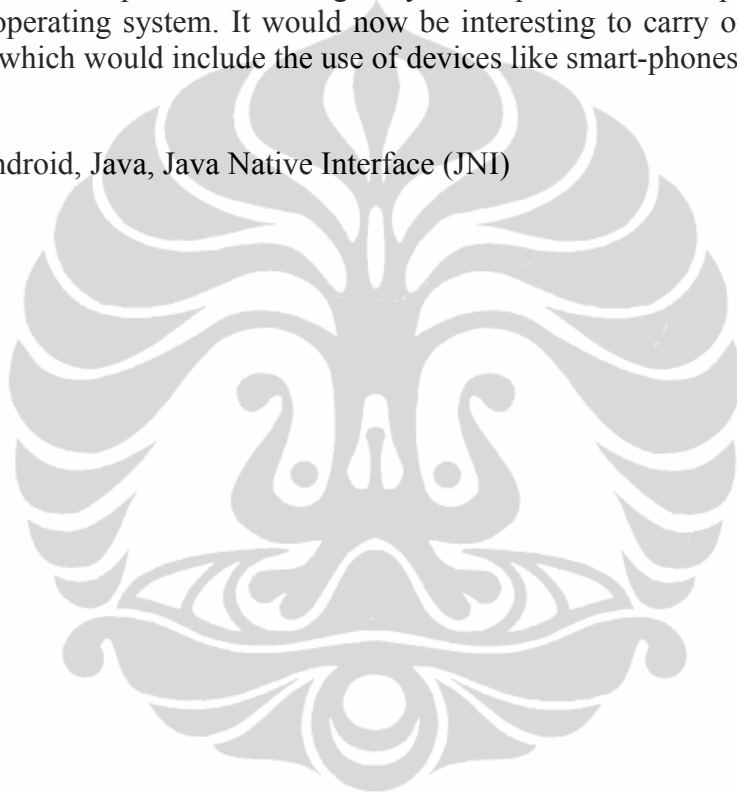
  
Fitri Wibowo

## ABSTRACT

Name : Fitri Wibowo  
Study Program : Electrical Engineering  
Title : Development of SAJE Project on Android Operating System

The group CASA of VALORIA have developed SAJE (System-Aware Java Environment) which is a framework dedicated to the observation of system resources in Java. This platform was originally developed to run on top of a Linux or Windows operating system. It would now be interesting to carry on a system like Android, which would include the use of devices like smart-phones.

Keywords: Android, Java, Java Native Interface (JNI)



## ABSTRAK

Nama : Fitri Wibowo  
Program Study : Teknik Elektro  
Judul : Development of SAJE Project on Android Operating System

Group CASA dari laboratorium riset VALORIA Universite de Bretagne-Sud mengembangkan sebuah framework yang bernama SAJE (System-Aware Java Environment). Framework ini didedikasikan untuk mengobservasi sumber daya yang ada dalam sebuah sistem operasi dengan menggunakan bahasa pemrograman Java. Pada awal pengembangannya, framework ini ditargetkan untuk berjalan di atas sistem operasi Linux atau Windows. Seiring dengan perkembangan pesat teknologi mobile saat ini, maka akan sangat menarik jika bisa menjalankannya pada sistem operasi yang berbeda seperti Android.

Kata kunci: Android, Java, Java Native Interface (JNI)



## TABLE OF CONTENTS

APPROVAL PAGE.....	ii
DECLARATION OF ORIGINALITY.....	iii
ACKNOWLEDGMENT.....	iv
DECLARATION OF PUBLICATION APPROVAL PAGE FOR ACADEMIC PURPOSE.....	v
ABSTRACT.....	vi
TABLE OF CONTENTS.....	viii
INDEX OF FIGURES.....	ix
INDEX OF TABLES.....	x
1. Introduction.....	1
1.1 Presentation of VALORIA.....	1
2. The SAJE project.....	2
2.1 Context.....	2
2.2 Motivations.....	3
2.3 The expected system.....	3
2.4 Known problem.....	3
2.5 Project organization.....	4
2.6 Development Environment.....	4
2.7.1 wireless-tools [4].....	9
2.7.2 net-tools [5].....	10
2.7.3 Busybox [6].....	10
2.7.4 SuperUser [7].....	11
3. Implementation.....	12
3.1 Setting up development Environment.....	12
3.2 Integrating wireless-tools into Android.....	14
3.3 Configuring the wireless interface in ad-hoc mode on Android.....	18
3.4 Porting SAJE project to Android.....	20
3.4.1 User Interface Prototype.....	21
3.4.2 Building SAJE shared library.....	22
3.4.3 CPU.....	24
3.4.4 Memory and Swap.....	26
3.4.5 Battery.....	28
3.4.6 Wi-Fi.....	30
3.5 Developing an application to control Wi-Fi.....	34
3.5.1 Device information section.....	35
3.5.2 Network configuration section.....	37
3.5.3 Profile management.....	38
3.5.4 Connection information.....	39
4. Conclusion.....	40
Bibliography.....	41
Appendix.....	42
Appendix 1 : Class diagram of Saje2Droid project.....	43
Appendix 2 : Class definition.....	44
Appendix 3 : Use case diagram.....	46

## INDEX OF FIGURES

Figure 1 : Project hierarchy in Eclipse IDE .....	15
Figure 2 : wireless-tools integration into project .....	15
Figure 3 : JNI output compilation on Android project .....	16
Figure 4 : SAJE integrated into Android project.....	21
Figure 5 : The UI prototype of MainActivity and the result on Android device ...	22
Figure 6 : Workflow diagram of Saje2Droid project .....	23
Figure 7 : Logcat informations of Saje2Droids shared library .....	24
Figure 8 : The UI prototype of CpuActivity and the result on Android device ...	26
Figure 9 (a) : Logcat information of memory report .....	26
Figure 9 (b) : Logcat information of swap report .....	26
Figure 10 : MemoryActivity and prototype the result on Android device .....	28
Figure 11 : BatteryActivity prototype and the result on Android device .....	30
Figure 12 : Logcat information of “NoSuchResource” exception .....	30
Figure 13 : WifiActivity prototype and the result on Android device .....	34
Figure 14 (a): Method selection for kernel module / wireless name .....	36
Figure 14 (b): WiFi mode selections dialog .....	36
Figure 14 (c): Channel selections dialog .....	36
Figure 15 : Typical input dialog uses in network configuration section .....	37
Figure 16 : Profile management menus .....	38
Figure 17 : WifiRunActivity .....	39

## INDEX OF TABLES

Table 1 : Associated permission uses by Saje2Droid.....	33
Table 2 : Profile management function in Saje2Droid.....	38



# 1. INTRODUCTION

## 1.1 Presentation of VALORIA

VALORIA is the computer science laboratory at *Université de Bretagne Sud* (UBS). It develops research activities in the scope of Ambient Intelligence (AmI), addressing research and development topics along three complementary research activities:

- **Interaction and Intelligence:** This first activity aims at providing end-users with technological, innovative means for greater user-friendliness, more efficient services support and user-empowerment, while contributing to user-friendly, dependable, adaptive and non-intrusive hardware/software environments.
- **Software Architecture:** The second activity is dedicated to architecting/refining, testing/refactoring and maintaining/evolving dynamic, distributed, mobile and context-aware systems considered as the background support to ambient computing.
- **Middleware:** The third activity focuses on providing middleware for distributed mobile and communicating systems as a support to ubiquitous and pervasive computing.

VALORIA is structured in four teams:

1. ARCHWARE (software architectures);
2. CASA (platforms for mobile computing);
3. SEASIDE (SEarch, Analyze, Simulate and Interact with Data Ecosystems);
4. RIMH (robotics and multimodal interactions towards disability).

## 2. THE SAJE PROJECT

### 2.1 Context

SAJE (System-Aware Java Environment) is a framework dedicated to the monitoring of system resources in Java. It makes it possible to model system resources (CPU, network interfaces, etc.) as Java objects, and it provides facilities to monitoring these resources through the corresponding objects. It thus offers useful functionalities to design and implement adaptive systems or resource monitoring systems for instance. SAJE is distributed under the GNU General Public License

In the current version of SAJE, the following resources can be observed:

- processor: model, speed, cache size, etc.
- memory: total size, available size, etc.
- swap memory: total size, available size, etc.
- networking interfaces: type (Loopback, Ethernet, WiFi, PPP, etc.), number of bytes and IP packets sent and received, etc.
- battery: management type (APM), loading state, lifetime, etc.

SAJE use Java Native Interface that allows Java to interact with code written in another programming language. The use of JNI will give some advantages, such as :

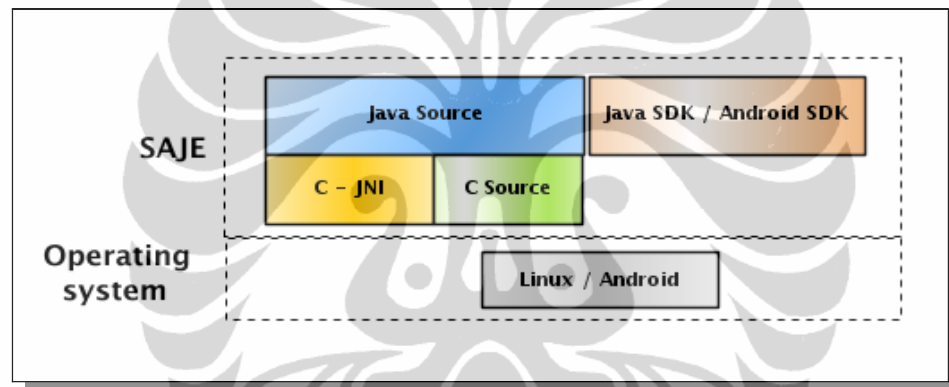
- Code re-usability: Reuse existing/legacy code with Java (mostly C/C++)
- Performance: Native code used to be up to 20 times faster than Java, when running in interpreted mode
- Allow Java to tap into low level O/S, H/W routines

## 2.2 Motivations

This platform was originally developed to run on top of a Linux or Windows operating system. It would now be interesting to carry on a system like Android, which would include the use of devices like smart-phones.

Android operating system is based on the Linux kernel, but the user space is built at top of Dalvik, a Google-designed custom JVM (Java virtual machine). This will give us a big opportunity to porting SAJE onto Android platform.

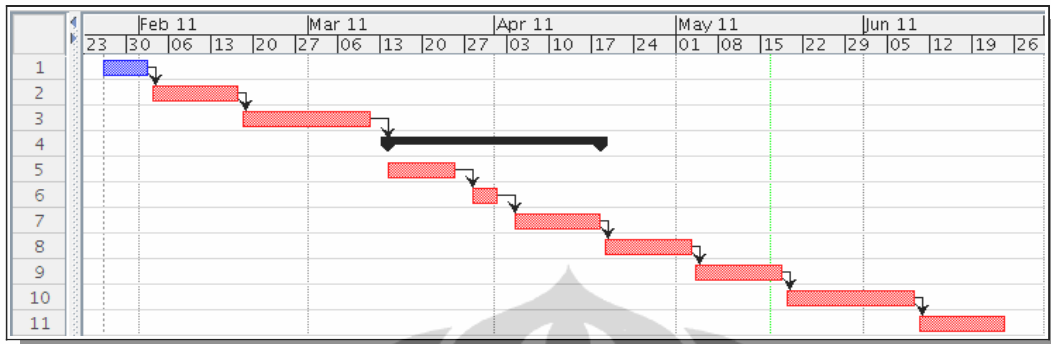
## 2.3 The expected system



## 2.4 Known problem

- Android does not use APM for the battery management.
- Android phone does not support wireless ad-hoc communication mode.
- Android's root permissions limitation.

## 2.5 Project organization



### Planning

1. Setting up development environment : 1 week.
2. Integrating wireless-tools to Android : 2 weeks.
3. Configuring wireless interface in ad-hoc on Android: 3 weeks.
4. Porting SAJE to Android: (5 weeks)
5. Porting memory & swap , cpu : 2 weeks
6. Battery: 1 week
7. Interface Wi-Fi: 2 weeks
8. Installation procedure with GNU autotools: 2 weeks
9. User manual: 2 weeks
10. Developing application to control Wi-Fi: 3 weeks
11. Project report: 2 weeks

## 2.6 Development Environment

### Eclipse IDE

Eclipse IDE is an Integrated development that come from Eclipse open source community, whose projects are focused on building an open development platform including extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. We use Eclipse in order to develop SAJE project for the Android platform. This IDE offers many advantages.

Indeed, Eclipse IDE provides a plugin for developing an application on the Android platform. This plugin is ADT (Android development tools). In addition, Eclipse IDE also has plugins to develop JNI applications by using Eclipse CDT (C/C++ Development Tools). This IDE provides a lot of versioning tools integration, and one of them is Subversive that uses for SVN.

### **ADT (Android Development Tools)**

Android Development Tools (ADT) is a plugin for the Eclipse IDE that is designed to give a powerful, integrated environment in which to build Android applications.

ADT extends the capabilities of Eclipse to let us quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug an application using the Android SDK tools, and even export signed (or unsigned) .apk files in order to distribute our application.

Developing in Eclipse with ADT is highly recommended because it is the fastest way to get started. With the guided project setup it provides, as well as tools integration, custom XML editors, and debug output pane, ADT gives us an incredible boost in developing Android applications.

The current version of ADT is v11.0.0 which is designed for use with Android SDK Tools r11. Eclipse and the ADT plugin provide GUIs and wizards to create all three types of projects (Android project, Library project, and Test project).

An Android project contains all of the files and resources that are needed to build a project into an .apk file for installation. We need to create an Android project for any application that we want to eventually install on a device. We can also designate an Android project as a library project, which allows it to be shared with other projects that depend on it. Once an Android project is designated as a library



project, it cannot be installed onto a device.

### **CDT (C/C++ Development tools)**

The CDT plugin provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform. Features include: support for project creation and managed build for various toolchains, standard make build, source navigation, various source knowledge tools, such as type hierarchy, call graph, include browser, macro definition browser, code editor with syntax highlighting, folding and hyperlink navigation, source code refactoring and code generation, visual debugging tools, including memory, registers, and disassembly viewers.

As mentioned in section 2.1, the project SAJE uses JNI which allows Java to communicate with low-level routines of OS, therefore the development of SAJE for Android platform require additional applications: the Android NDK (Native Development Kit).

The Android NDK is a tool set that lets us embed components that make use of native code in the Android applications. Android applications run in the Dalvik virtual machine. The NDK allows us to implement parts of our applications using native-code languages such as C and C++. This can provide benefits to certain classes of applications, in the form of reuse of existing code and in some cases increased speed.

The Android NDK provides:

- A set of tools and build files used to generate native code libraries from C and C++ sources
- A way to embed the corresponding native libraries into an application package file (.apk) that can be deployed on Android devices
- A set of native system headers and libraries that will be supported in all future versions of the Android platform, starting from Android 1.5.

Applications that use native activities must be run on Android 2.3 or later.

- Documentation, samples, and tutorials

The latest release of the NDK supports these ARM instruction sets:

- ARMv5TE (including Thumb-1 instructions)
- ARMv7-A (including Thumb-2 and VFPv3-D16 instructions, with optional support for NEON/VFPv3-D32 instructions)

The future releases of the NDK will include x86 instructions

The contents of Android NDK are:

- The NDK APIs, documentation, and sample applications that help us to write a native code.
- a set of cross-toolchains (compilers, linkers, etc..) that can generate native ARM binaries on Linux, OS X, and Windows (with Cygwin) platforms.

It provides a set of system headers for stable native APIs that are guaranteed to be supported in all later releases of the platform, such as:

- libc (C library) headers
- libm (math library) headers
- JNI interface headers
- libz (Zlib compression) headers
- liblog (Android logging) header
- OpenGL ES 1.1 and OpenGL ES 2.0 (3D graphics libraries) headers
- libjnigraphics (Pixel buffer access) header (for Android 2.2 and above).
- A Minimal set of headers for C++ support
- OpenSL ES native audio libraries
- Android native application APIS

## **Pencil**

In order to create the user interface of Saje2Droid, we use the Pencil application<sup>[1]</sup>. Pencil is an open source applications for prototyping the graphical user interface, including both desktop base and mobile base application. It has two modes of installation, first; in the form of Firefox browser extension. Second mode, standalone application that is executed with the help of xulrunner<sup>[2]</sup>. As well as eclipse, Pencil is also extensible, in which to create a prototype user interface of android application, we use plugins called android-ui-tools<sup>[3]</sup> which is available for free.

## **android-ui-tools**

This utilities help in the design and development of Android application user interfaces. This library currently consists of three individual tools for designers and developers:

1. UI Prototyping Stencils

A set of stencils for the Pencil GUI prototyping tool, which is available as an add-on for Firefox or as a standalone download. Pencil 1.2 and Firefox 3.5 or later are required.

2. Android Asset Studio

A web-based set of tools for generating graphics and other assets that would eventually be in an Android application's res/ directory. Currently available asset generators area available for:

- Launcher icons
- Menu icons
- Tab icons
- Notification icons
- Support for creation of XML resources and nine-patches is planned for a future release.

### 3. Android Icon Templates

A set of Photoshop icon templates that follow the icon design guidelines, complementing the official Android Icon Templates Pack.2.7 Required applications

In the beginning of its development, SAJE project is designed to run only on Linux operating system. Therefore, there are some applications or additional libraries in Linux that are required by SAJE, such as the wireless tools which is the back-end of the the wireless device configuration, and net-tools which is a collection of standard tools to configure network in Linux operating system. Additionally, to make adjustments with the Android operating system on the porting process, it is also needed some additional applications to be exist on the Android operating system . Among of these are: administrator access (root) on the Android operating system level that can be configured by SuperUser application, and also a collection of standard Linux commands that are available in the Busybox application. Both of these applications is available in Android market.

#### 2.7.1 wireless-tools <sup>[4]</sup>

The Linux Wireless Extension and the Wireless Tools are an Open Source project sponsored by Hewlett Packard since 1996. The Wireless Extension (WE) is a generic API allowing a driver to expose to the user space configuration and statistics specific to common Wireless LANs. The advantage of it is that a single set of tool can support all the variations of Wireless LANs, regardless of their type (as long as the driver support Wireless Extension). Another advantage is these parameters may be changed on the fly without restarting the driver (or Linux).

The Wireless Tools (WT) is a set of tools allowing to manipulate the Wireless Extensions. They use a textual interface and are rather crude, but aim to support the full Wireless Extension. There are many other tools you can use with Wireless Extensions, however Wireless Tools is the reference implementation.

- iwconfig manipulate the basic wireless parameters
- iwlist allow to initiate scanning and list frequencies, bit-rates, encryption keys...
- iwspy allow to get per node link quality
- iwpriv allow to manipulate the Wireless Extensions specific to a driver (private)
- ifrename allow to name interfaces based on various static criteria

Our objectif is to run this set of wireless tools on Android operating systems.

### 2.7.2 net-tools <sup>[5]</sup>

Android operating system comes with some standard Linux command like ifconfig. But this version of ifconfig is a modified version that made for fit to run on Android operating system, which mean that its size is optimized. Since SAJE project need some output of the original command of Linux's ifconfig, we decided to try to compile original net-tools on Android operating system so SAJE could run perfectly on Android.

The Linux's net-tools is a collection of programs that form the base set of the NET-3 networking distribution for the Linux operating system. This package includes arp, hostname, ifconfig, ipmaddr, iptunnel, mii-tool, nameif, netstat, plipconfig, rarp, route and slattach. We do not need to compile all of this binary, since SAJE just need the ifconfig command.

### 2.7.3 Busybox <sup>[6]</sup>

BusyBox is a software application that provides many standard Unix tools, much like the larger (but more capable) GNU Core Utilities. BusyBox is designed to be a small executable for use with the Linux kernel, which makes it ideal for use with embedded devices. It has been self-dubbed "The Swiss Army Knife of Embedded Linux".

#### 2.7.4 SuperUser <sup>[7]</sup>

The Superuser application is intended to grant and manage superuser rights for Android device. This application requires that we already have root access, or a custom recovery image to work.



## 3. IMPLEMENTATION

### 3.1 Setting up development Environment

#### Eclipse + plugins

The installation of Eclipse IDE can be done in two ways, first we can install it from a distributed package that come with the Linux distribution, or to install it from an archive package that could work on every Linux distribution as long as it matched with the processor architecture.

The standard Eclipse installation does not have any plugins mentioned above. So we need to install it manually one by one. First we need to add software site for each plugins to Eclipse. A software site is a collection of URL that contains Eclipse plugins. The URL for the plugins mentioned above (assuming that we use Eclipse helios version):

- ADT (Android Development Tools) - <https://dl-ssl.google.com/android/eclipse/>
- CDT (C/C++ Development Tools) - <http://download.eclipse.org/tools/cdt/releases/helios>
- Subversive - <http://download.eclipse.org/releases/helios>

The Subversive plugin is part of Helios update release, so once the software update catalog is updated the Subversive plugin should be listed in software site. Subversive installation includes Subversive plug-in and Subversive SVN Connectors plug-in. In order to start work with Subversive we should install both of them.

The installation process of Eclipse plugins can be done by selecting Help menu from Eclipse and then clicking the Install new software menu. This step will show an install window. From this window we can select the available software sites that we have defined in the previous process.

### Setting up ADT (Android Development Tools)

After we have successfully downloaded the ADT as described above, the next step is to modify ADT preferences in Eclipse to point to the Android SDK directory. The Android SDK location's settings is located in the preferences panel of Eclipse.

### Setting up CDT to use Android NDK

By default CDT does not have integration feature with Android NDK. However, the CDT plugins support the C/C++ Makefile project. We can anticipate this by setting up system environment which locate the Android NDK or set the parameters in the Makefile of the project. This is the code snippets in the Makefile that we use:

```
# Set this variable to the android-ndk location
ANDROID_NDK_BASE:=/home/user/Apps/android-ndk-r5b
PATH:=${PATH}:${ANDROID_NDK_BASE}

all:
    ndk-build

clean:
    ndk-build clean
```

So instead of using Java perspective in Eclipse IDE, we use C/C++ perspective to develop this project.

### Setting up Subversive and SVN CASA

To help us with the synchronization and project management, we use a versioning tool SVN (Subversion). Saje2droid project is uploaded to SVN Casa that located at "<https://www-valoria.univ-ubs.fr/svn/casa/devlpts/Saje2Droid>". The access to the SVN CASA is currently private which means only the members of the Laboratoire Valoria can access it.



To configure Subversive with SVN CASA, we have to define a SVN Repository by selecting menu File → New → Other in Eclipse. The next step we need to fill in the Repository address of SVN Casa, and provide the credential login for this repository. If the process is done successfully, we can explore the repository by selecting SVN repository perspective in Eclipse.

### **3.2 Integrating wireless-tools into Android.**

Based on the project planning, we need to know first that the wireless tools can run on the Android operating system. In this process we try to cross compile wireless tools into executable binary files that can be executed on the Android operating system with the help of toolchain / cross platform tools provided by Android NDK.

There are two ways to distribute a binary of the wireless tools that will be compiled. First, distributed it along with the Android operating system. If we want to use this way, we need the full source code of the Android operating system that we compiled with some options which is suitable with the target device architecture. In addition we must add the source code from the wireless tools into the Android source code tree (normally in android/system/extras/). The next step is to make Android.mk which is written to describe the sources configuration to the build system.

The Android.mk is a tiny fragment of GNU Makefile that will be parsed one or more times by the build system. This file contains several compilation options such as library/headers, and the result of compilation like executable, shared libraries, static libraries, etc. The output location of the binary/library wireless tools will be affected by compilations settings defined in Android.mk. For example, to build a single executable file, we have to define BUILD\_EXECUTABLE option. However to build a shared/static library, we need to define LOCAL\_MODULE option.

The default output of the compilation is in `out/target/product/generic/` (relative to the Android source code). The next step is uploading the compilation's result of wireless tools to Android device. This process can be done with the help of `adb` (Android Debug Bridge) command that include in the Android SDK package.

The second way to integrate wireless tools to the Android operating system is compiling it as a native interface of Android application. In this way we do not need the full source of Android, but as JNI application, we need Android NDK. As explained in the previous section, after creating a Makefile project in eclipse, we still have to create an `Android.mk` file, and some additional directory in the Eclipse projects. These directory are:

- **libs, obj** – that used for maintaining the output of compilation process
- **jni** – that will be contains the C code of JNI which include wireless-tools

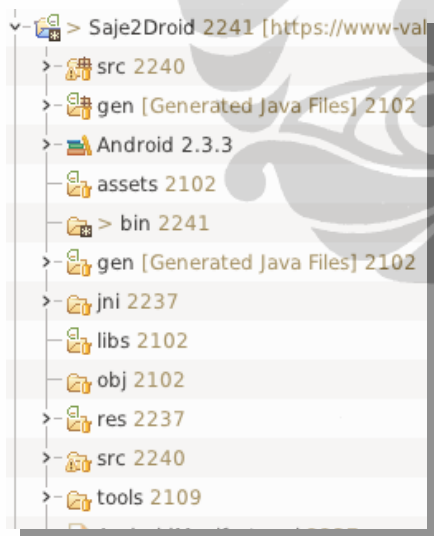


Figure 1 : Project hierarchy in Eclipse IDE

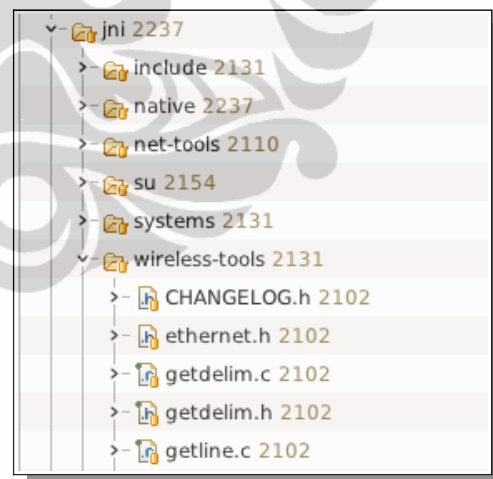


Figure 2 : wireless-tools integration into project

This is the code snippet from `Android.mk` used by wireless-tools to compile “iwconfig” binary. The result file name specified by `LOCAL_MODULE` variable.

```

include $(CLEAR_VARS)
LOCAL_SRC_FILES := iwconfig.c
LOCAL_MODULE := iwconfig_saje
LOCAL_STATIC_LIBRARIES := iwlib
LOCAL_CFLAGS := $(WT_INCS) $(WT_DEFS)
include $(BUILD_EXECUTABLE)

```

With this second way, the wireless-tools binary will be compiled along with the Android package. The default location of the Android package kit is inside "bin" directory, while for JNI objects and binaries are in the "libs/armeabi" directory.

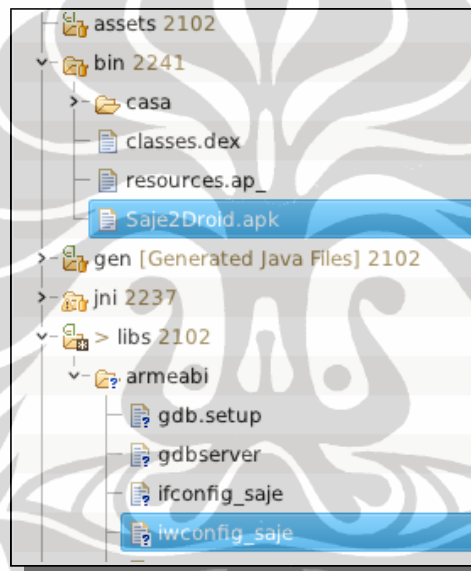


Figure 3 : JNI output compilation on Android project

When the android package kit is installed into the Android operating system, the binary files from C/C++ code is not installed automatically. To include the executable binary file into the Android package can be done by copying it into the "res/raw" folder.

The next process is extracting the raw file from the Android package kit. Android provides a method called `openRawResource` from the class `Resource` with ID of resource as parameter. This method will return an `InputStream` of raw file that we specified by it's ID<sup>[8]</sup>. From this input stream we can put it to `OutputStream` line by

line to get the original raw file. Here is the code snippet that we use to copy iwconfig\_saje binary to Android package kit, and then extracting it automatically when the Saje2Droid application is executed.

```
File outFile = new
File("/data/data/casa.saje/bin/iwconfig_saje");
InputStream is =
this.getResources().openRawResource(R.raw.iwconfig_saje);
byte buf[] = new byte[1024];
int len;
try {
    OutputStream out = new FileOutputStream(outFile);
    while ((len = is.read(buf)) > 0) {
        out.write(buf, 0, len);
    }
    out.close();
    is.close();
} catch (IOException e) {
    return "Couldn't install file - " + filename + "!";
}
```

We set the location of wireless-tools binary in "bin" directory inside /data/data/casa.saje which is the default data path of android application. Once the file is copied from Android resource to the system we need to change its permission to executable. There was a little problem while testing wireless-tools binary in Android OS, since the Android emulator does not support yet WiFi device emulation<sup>[9]</sup>, we need to test it on "real" Android device. These are the output of iwconfig command tested on rooted HTC Desire and android device emulator :

HTC Desire :

```
# ./iwconfig_saje
lo          no wireless extensions.
dummy0     no wireless extensions.
rmnet0     no wireless extensions.
rmnet1     no wireless extensions.
rmnet2     no wireless extensions.
usb0       no wireless extensions.
sit0       no wireless extensions.
```

```

ip6tnl0  no wireless extensions.
eth0     IEEE 802.11-DS  ESSID:""  Nickname:""
         Mode:Managed  Frequency:2.412 GHz  Access Point:

Not-Associated
         Bit Rate:54 Mb/s   Tx-Power:32 dBm
         Retry min limit:7   RTS thr:off   Fragment thr:off
         Encryption key:off
         Power Managementmode:All packets received
         Link Quality=5/5   Signal level=0 dBm   Noise
level=0 dBm
         Rx invalid nwid:0   Rx invalid crypt:0   Rx invalid
frag:0
         Tx excessive retries:0   Invalid misc:0   Missed
beacon:0

```

Android device emulator :

```

# ./iwconfig_saje
lo       no wireless extensions.
eth0     no wireless extensions.
tunl0    no wireless extensions.
gre0     no wireless extensions.

```

### 3.3 Configuring the wireless interface in ad-hoc mode on Android

After integrating successfully the wireless-tools into Android operating system , the next step is to configure the WiFi interface on Android by using the wireless tools. There are several processes that needed to be done to ensure that the wireless tools can be run on the Android operating system. We know that wireless tools requires the wireless extensions to be enabled in Android kernel. So we must check first whether the kernel of Android has wireless extensions enabled in its kernel.

First method to check wireless extension on Android operating system is by checking the variable CONFIG\_WIRELESS\_EXT=y or CONFIG\_NET\_RADIO=y (depends on the kernel version) inside the kernel

configuration. The second method is verifying the existence of the file `/proc/net/wireless`, since by default this file will be created when the Linux kernel has wireless extensions support <sup>[10]</sup>.

When the wireless extension is enabled in the kernel of the Android operating system, the wireless device configuration can be done with the wireless tools which have been integrated before. To avoid crash configuration with the default wireless management tools on Android, it must be noticed earlier that the `wpa_supplicant` process is terminated. After that we need to know the kernel module for the wireless interface so it can be loaded manually.

To change the wireless mode settings is using the `iwonfig` command, with an option "mode". In certain cases, the channel option must be specified. Since this command is configuring the hardware settings, this requires administrator (root) access on the Android operating system side. At this stage, to verify the connectivity on an ad-hoc networks in Android operating system, we can use the command ICMP Echo Request (ping). At the end of the project, testing the connectivity is done by including client server applications using UDP protocol.

We did an ad-hoc test between Android devices and a personal computer. The result shows that the ICMP echo request works successfully. There is no big problem in this test except that some Android device has power management turned ON by default, to solve this issue, we have to modify its power management by turning it OFF. This is the example ad-hoc configuration step that we use on HTC Desire running on Android rooted Android 2.2 :

```
# killall wpa_supplicant
# insmod /system/lib/modules/bcm4329.ko
# ./iwlist_saje eth0 scan
eth0      Scan completed :
          Cell 01 - Address: BE:D0:3D:6C:F8:1A
                   ESSID:"ika"
                   Mode:Ad-Hoc
```

```

Frequency:2.412 GHz (Channel 1)
Quality:5/5 Signal level:-43 dBm Noise
Level:-92 dBm
Encryption key:off
Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s;
6 Mb/s
9 Mb/s; 12 Mb/s; 18 Mb/s; 24 Mb/s;
36 Mb/s
48 Mb/s; 54 Mb/s

# ./iwconfig_saje eth0 power off
# ifconfig eth0 down
# ./iwconfig_saje eth0 mode ad-hoc channel 1
# ifconfig eth0 192.168.0.3 up
# ./iwconfig_saje eth0 essid ika
# ping -c1 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=128 time=7.59 ms
--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 7.599/7.599/7.599/0.000 ms

```

### 3.4 Porting SAJE project to Android

In this task, we start to integrate SAJE to the Android project Saje2Droid. We use the latest revision of SAJE. This version of SAJE contains some improved code for Android like system command execution and better wireless device detection. Since we need to modify some code in SAJE to match with Android OS requirement, we included the SAJE source code tree to the project and not just as JAR library. To integrate a Java project like SAJE to Android project, we just need to copy it inside “src” directory.

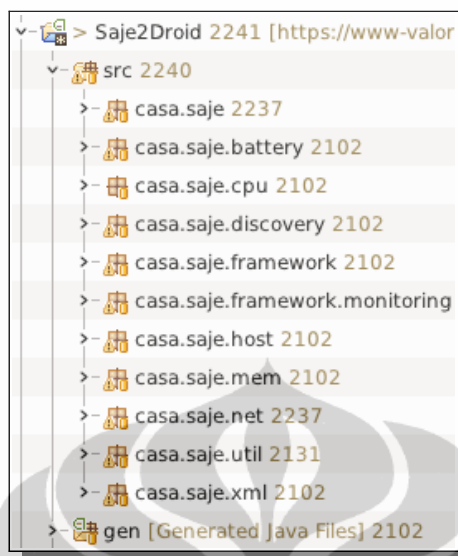


Figure 4 : SAJE integrated into Android project

### 3.4.1 User Interface Prototype

The important step when developing an Android application is prototyping the user interface. We did the prototype with Pencil and android-ui-utils. The first prototype created was using TabView activity, but at the end we decided to use ListView activity since it is more simple and like common Android applications so it improve the user experience and interaction.

There are 6 Android activities used in this project. The first activity is MainActivity that will showed up when the Application is executed and within this activity, the other activity can be started. The other activities are CpuActivity that contains CPU information, MemoryAcitivity that contains memory and swap information, BatteryActivity that contains current Battery status, WifiActivity contains wireless and network configurations, and the last Activity is WifiRunActivity that will be started when all WiFi settings in WifiActivity is applied successfully and it will show a notification and connections information like SSID and IP of the WiFi interface:





Figure 5 : The UI prototype of MainActivity and the result on Android device

### 3.4.2 Building SAJE shared library

A shared library is a binary file that contains a set of callable C functions. In this case, we tried to make a function collections from C sources of SAJE project so Android application may load and unload this shared libraries at will. The C sources of SAJE project itself depends on wireless tools headers for its compilation. This figure shows the workflow diagram of Saje2droid projects.

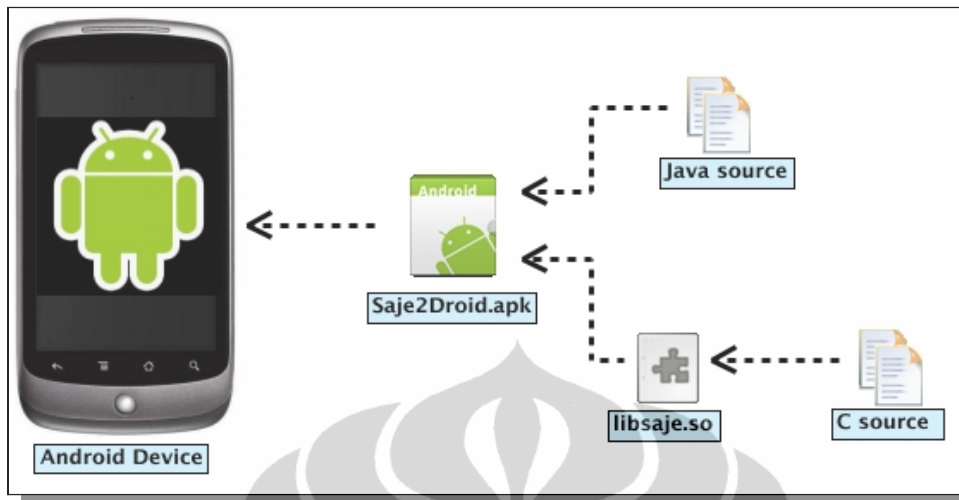


Figure 6 : Work-flow diagram of Saje2Droid project

The Android.mk configure the build parameters for the C source. To build a shared library libsaje.so we specified a variable \$(BUILD\_SHARED\_LIBRARY) at the end of our configuration. This code snippet shows the build configuration Android.mk of Saje2Droid

```
## libsaje
include $(CLEAR_VARS)
LOCAL_PATH := $(BASE_PATH)
LOCAL_SRC_FILES := $(SAJE_SRC_FILES)
LOCAL_C_INCLUDES += \
    $(LOCAL_PATH)/include \
    $(LOCAL_PATH)/wireless-tools
LOCAL_STATIC_LIBRARIES := iwlib
LOCAL_MODULE := saje
include $(BUILD_SHARED_LIBRARY)
```

The above code will tell Android build system to generate a shared library from the source file defined in LOCAL\_SRC\_FILES variable. Since Saje2Droid depends on wireless tools header , we specified it with LOCAL\_C\_INCLUDES variable and LOCAL\_STATIC\_LIBRARIES. Once the project compilations is done the libsaje.so will be founded in output directory (libs/armeabi/libsaje.so) and will be included automatically to android package kit as application dependencies.

When the Saje2Droid application is installed on Android OS, the shared library should be copied automatically to `{APPLICATION_DATA_PATH}/{PACKAGE_NAME}/lib`. The default data path of Android application is in `/data/data`. So we can find `libsaje.so` in `/data/data/casa.saje/lib/libsaje.so`. This shared library will be loaded automatically when the application is started. The Android's logcat will show this information.

```

ActivityManager Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.cat
ActivityManager Start proc casa.saje for activity casa.saje/.MainActivity: pid=315 uid=10
dalvikvm Trying to load lib /data/data/casa.saje/lib/libsaje.so 0x40514160
dalvikvm Added shared lib /data/data/casa.saje/lib/libsaje.so 0x40514160
dalvikvm No JNI_OnLoad found in /data/data/casa.saje/lib/libsaje.so 0x40514160, s
szipinf Initializing inflate state
szipinf Initializing zlib to inflate
ActivityManager Launch timeout has expired, giving up wake lock!
ActivityManager Displayed casa.saje/.MainActivity: +10s6ms

```

Figure 7 : Logcat informations of Saje2Droids shared library

### 3.4.3 CPU

In this part, we start to test the CPU codes of SAJE into Android OS. The original code does not work on Android because we found that the Androids `/proc/cpuinfo` has different structure with that in Linux operating system. So we modified the CPU code in SAJE to match with the structure of `/proc/cpuinfo` on Android operating system.

The modified files are **`jni/system/linux/i386/cpuinfos.c`**, **`jni/include/system/cpuinfo.h`**, **`jni/native/CpuModel.c`** and **`src/casa/saje/cpu/CPUModel.java`**. For the current version of Saje2Droid, we only use some informations from Android `cpuinfo` such as Processor, BogoMIPS, Features, CPU architecture and Hardware. This is an example of `/proc/cpuinfo` structure on Android OS and Linux OS.

Android :

```
# cat /proc/cpuinfo
Processor      : ARMv6-compatible processor rev 2 (v6l)
BogoMIPS      : 460.06
Features       : swp half thumb fastmult edsp java
CPU implementer : 0x41
CPU architecture: 6TEJ
CPU variant    : 0x1
CPU part       : 0xb36
CPU revision   : 2
Hardware       : trout
Revision       : 0080
Serial         : 0000000000000000
```

Linux :

```
# cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 42
model name    : Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz
stepping      : 7
cpu MHz       : 800.000
cache size    : 3072 KB
physical id   : 0
siblings      : 4
core id       : 0
cpu cores     : 2
...
```

The UI prototype of CpuActivity and the result on Android device :



Figure 8 : The UI prototype of CpuActivity and the result on Android device

#### 3.4.4 Memory and Swap

The original memory and swap code of SAJE was expected to work on Android device since there is no different structure on `/proc/meminfo` between Linux and Android OS. But there was a problem when we tried to show the result of MemoryActivity. The problem was the values of memory and swap in the result is not correct. This is the output from the logcat command on Android device :

```

at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:626)
at dalvik.system.NativeStart.main(Native Method)
MemoryReport= (sourceId= ResourceId(id= 'Resource#5') totalMemory= -1345042572
NET_RSSI_IND [message_id=0x1E len=3]:
* ** 1E 32 5E : ..2^

```

Figure 9 (a) : Logcat information of memory report

```

r pokeWakelock(5000)
set_light_keyboard 0
Window already focused, ignoring focus gain of: com.android.internal.view
SwapReport =(SourceId = ResourceId(id= 'Resource#6') Swap Size = 2306792
Failed to initiate AP scan.
SwapReport =(SourceId = ResourceId(id= 'Resource#7') Swap Size = 2337184
NET_RSSI_IND [message_id=0x1E len=3]:

```

Figure 9 (b) : Logcat information of swap report

As we can see in the figure above, the MemoryActivity reports that the total memory on the current system is in minus values and it reports that the current Android OS has certain values of swap. However it does not match with the contents of /proc/meminfo. These are the contents of /proc/meminfo from the same Android device :

```

# cat /proc/meminfo
MemTotal:      256556 kB
MemFree:       123740 kB
Buffers:        12 kB
Cached:        63092 kB
SwapCached:    0 kB
...
Mlocked:       0 kB
SwapTotal:     0 kB
SwapFree:      0 kB
Dirty:         0 kB
Writeback:     0 kB

```

After we investigated the memory and swap code in SAJE, we found that it uses C preprocessor directives `#ifdef LINUX_KERNEL_2_6` or `LINUX_KERNEL_2_4`. This cause the incorrect reports on memory and swap Activity, even the compilation process was done successfully. The problem solved by adding a new preprocessor directives called `ANDROID`, and then specify it in `LOCAL_CFLAGS` variable of the `Android.mk`. This is the code snippet of `Android.mk` that we used :

```

...
LOCAL_CFLAGS := -g -DANDROID
LOCAL_STATIC_LIBRARIES := iwlib \
                        libc
LOCAL_MODULE := saje
...

```

The UI prototype of MemoryActivity and the result on Android device :



Figure 10 : The UI prototype of MemoryActivity and the result on Android device

### 3.4.5 Battery

The current version of SAJE uses APM (Advance Power Management) library to manage battery while running on Linux operating system. Since Android does not use APM to manage its battery, we can not use SAJE battery code on Android operating system. However Android still provide sysfs that exports information about devices and drivers from the kernel device including battery information that located in `/sys/class/power_supply/battery`.

For the current version of Saje2Droid, we have not implemented yet the battery information from sysfs of Android operating system. However Android already

have a class located in `android.os.BatteryManager` that provides the information about the battery of the current Android operating system. The `BatteryManager` class contains string and constants used for values in the `ACTION_BATTERY_CHANGED` Intent.

To listen the `ACTION_BATTERY_CHANGED` Intent, we need to register a `BroadcastReceiver` (`android.content.BroadcastReceiver`) when the Intent is fired. The receiver has a function called `onReceive()` that called when the `BroadcastReceiver` is receiving an Intent broadcast.

The current version of Saje2Droid provides some battery informations such as level of battery in percent, voltage, temperature, technology used, status of battery (charging, dis-charging, not-charging, full), plug-type (unplugged, AC power, USB power, AC/USB power, unknown) and health quality (good, over heat, dead, over voltage, unspecified failure, unknown).

The UI prototype of `BatteryActivity` and the result on Android device :

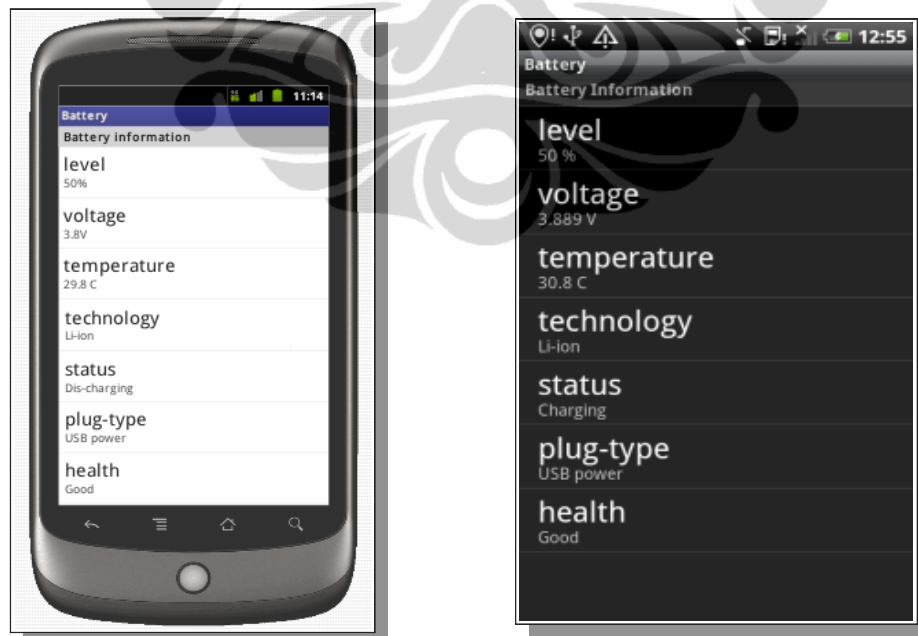


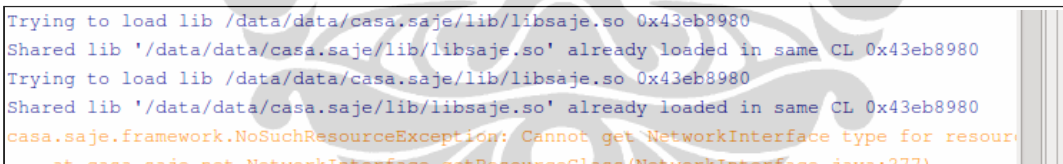
Figure 11 : The UI prototype of `BatteryActivity` and the result on Android device



### 3.4.6 Wi-Fi

The last part of SAJE project that we tried to port to Android OS is WiFi code. SAJE has network devices discovery support that located in `NetworkInterfaceDiscovery` class located inside `casa.saje.discovery` package that works perfectly on the Linux operating system. But we have not use this feature on the first attempt of porting. We have specified manually the interface name of WiFi interface on Android by investigating it on the Android shell, and then create an instance of `NetworkInterface` class of this interface.

There was a problem that occurred on first attempt of porting process. This problem occurred when the object of `NetworkInterface` is instantiated, SAJE will throw "NoSuchResource" exception with additional message "Cannot get NetworkInterface type for resource named 'eth0!'". In this test, we use Android Cyanogenmod that running on HTC Dream device that does not have wireless extension enabled in its kernel. This is the detailed message showed on Android logcat:



```
Trying to load lib /data/data/casa.saje/lib/libsaje.so 0x43eb8980
Shared lib '/data/data/casa.saje/lib/libsaje.so' already loaded in same CL 0x43eb8980
Trying to load lib /data/data/casa.saje/lib/libsaje.so 0x43eb8980
Shared lib '/data/data/casa.saje/lib/libsaje.so' already loaded in same CL 0x43eb8980
casa.saje.framework.NoSuchResourceException: Cannot get NetworkInterface type for resource named 'eth0!'
at casa.saje.framework.NoSuchResourceException.<init>(NetworkInterfaceDiscovery.java:277)
```

Figure 12 : Logcat information of "NoSuchResource" exception

The result is the same when we compared to the Android devices that supports wireless extension. The Android version tested was Froyo (2.2) that running on Nokia N900 device. This version of Android is known to have wireless extension support in its kernel. This error was caused by `getResourceClass` function in `NetworkInterface` class that can not get the type of the network interface given. This function should return the appropriate subclass of `NetworkInterface` corresponding to the type of the given network interface's name by calling a native function `getType()`.

The function `getType()` will return an integer value of the network interface type that is defined before as constant. For the wireless network interface it should be 71 defined as `IFTYPE_WIRELESS`. The native code will verify the wireless interface by calling `is_wireless()` function located in `wirelessif.c` and the problem was this method could not detect WiFi interface correctly. Since SAJE revision 1978 this bug is corrected.

However the wireless network device initialization error was not solved yet. We try to debug the application and we found that SAJE uses “ifconfig” command to find network device information by parsing in its output. The Android's “ifconfig” is a modified version of Linux's “ifconfig”, so it does not provide the information that needed by SAJE.

To solve this problem, we did a cross compile to the original ifconfig's source by integrating net-tools source code to JNI code in Saje2Droid project's tree. We specified to compile “ifconfig” command line only since net-tools comes with a collection of standard tools for network configuration. This is the code snippet of `Android.mk` for compiling net-tools's “ifconfig” command for Android:

```
# build ifconfig-saje executable
include $(CLEAR_VARS)
LOCAL_PATH := $(NT_PATH)
LOCAL_SRC_FILES:= ifconfig.c
LOCAL_C_INCLUDES += $(LOCAL_PATH)/include \
                   $(LOCAL_PATH)/lib

LOCAL_STATIC_LIBRARIES := netlib
LOCAL_MODULE := ifconfig_saje
include $(BUILD_EXECUTABLE)
```

Wireless device can be initiated successfully after the compilation of “ifconfig”. The next step was getting WiFi device information such as SSID name, channel

used by device, WiFi mode (ad-hoc, managed) through the native code in shared library. The current version of Saje2Droid can implement the `getEssid()` function from `WirelessInterface` class but for the others function we have not implemented it yet.

The last work on porting wireless interface code is to set the WiFi parameters with SAJE. The current version of SAJE provides several functions to set the WiFi parameters through the wireless tools library. This function includes setting the SSID, WiFi channel, WiFi mode, and frequencies used by the WiFi device. This function is known to work on Linux operating system with administrator (root) privileges because wireless tools uses “`ioctl`” function to set and get network parameters on wireless device[3]. This function induces a problem on Android device because Android would never run an application as root.

Android uses associated permission mechanism that allow an application to do a specific operations. This permission is defined in `AndroidManifest.xml` that can be found inside each Android project within tag `<uses-permission android:name="REQUIRED_PERMISSION"/>`. The `REQUIRED_PERMISSION` is a constant string that provided by Android system. The table below shows the Android permissions used by current version of Saje2Droid:

Constant string	Associated permissions
HARDWARE_TEST	Allows access to hardware peripherals.
ACCESS_WIFI_STATE	Allows applications to access information about Wi-Fi networks
CHANGE_WIFI_STATE	Allows applications to change Wi-Fi connectivity state
UPDATE_DEVICE_STATES	Allows an application to update device statistics.
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming
BATTERY_STATS	Allows an application to collect battery statistics

Table 1 : Associated permission uses by Saje2Droid

The issue on Wi-Fi parameters setting is still not solved yet even Android permissions is specified in AndroidManifest.xml. The solutions that currently proposed to set Wi-Fi parameter is using binary file of wireless tools such as “iwconfig”, “iwlist” that executed by root user on Android operating system. Unfortunately, the default Android operating system does not allow root access on it's shell. Due to this limitation, the current solutions will work only on rooted Android phone.

### 3.5 Developing an application to control Wi-Fi

The last step of this project is developing an Android activity that can control the Wi-Fi interface in Android operating system. This features uses wireless tools as back-end of wireless configuration and depends on several standard Linux tool such as iproute, net-tools to set the network parameters. This figure below shows the prototype of Android activity that control the Wi-Fi and the result on Android device:



Figure 13 : The UI prototype of WifiActivity and the result on Android device

This Android activity is defined by WifiActivity class. Like the others Android activity in Saje2Droid project, this class uses simple ListView as main interface. For each list, it consists of 3 Android's TextView objects. The first TextView is the title of the header in each section. As we can see in the figure above, we separate the Wi-Fi device settings with the network configuration into 2 separate sections. The second TextView is the title of the list that we called parent TextView. The parent TextView is marked by using a bigger font size. The last TextView is

located below the parent TextView that we called child TextView. This TextView uses a smaller font size. The child view is used for displaying certain values, so we set it to dynamically changeable by the application.

All of these static user interface settings saved in a XML file that defined the layout of Android activity. This XML must be stored in “res/layout” directory. The static string values can be saved also in a XML file called strings.xml located inside “res/values” directory of the Android project. This listview is configured to click-able so it receives an events click/tap it will show dialog that ask user input.

### 3.5.1 Device information section

In the device information sections, we currently implement kernel module configuration, network interface name configuration, ssid of Wi-Fi device, Wi-Fi mode, and channel. We have improved the kernel module configuration by adding an automatic kernel module name detection. In the wireless interface name configuration, we implement an automatic detection using the NetworkInterfaceDiscovery class of SAJE. The class will return a set of string of the network device founded in the device. And then for each device we initiate it to a NetworkInterface instance. After it, we can verify a wireless interface by calling a native function getType.

This features is known work on certain Android devices. Since Android is an open source operating system hence there are a lot of vendor that uses their own version of Android means that they may have modified some default parameters of Android operating system, so this features may not work. In this situation, the users still can use the manual input of wireless kernel module and network interface name.



Figure 14 (a): Method selection for kernel module / wireless interface name configuration, (b): WiFi mode selections dialog, (c): channel selections dialog

For the ssid setting, it currently supports manual input only. It will be interesting to implement wireless network cell detection in the future work since SAJE already had a native function to get a list of wireless network cell. In the Wi-Fi

mode settings, we provide two options. First is ad-hoc and the second is managed. So far, we have not work a lot in the managed Wi-Fi mode , since this projects is focusing on an ad-hoc network between Android device. The wireless channel configuration currently supports standard European model that allowing channels 1 through 13.

### 3.5.2 Network configuration section

The second section of Listview in WifiActivity provides standard network configuration. In the current version of Saje2Droid, we can configure IPv4 address, netmask, gateway, DNS server, and IPv6. The IPv6 implementation is still under development since not all of Android device enable the IPv6 supports. In an ad-hoc network, we do not have to fill in the gateway and DNS server configuration if the device is not connected to a network router. So when the value of gateway and DNS server is empty, the application will not update the routing table and DNS server address of the Android operating system. This figure below shows a dialog to input an IPv4 address:

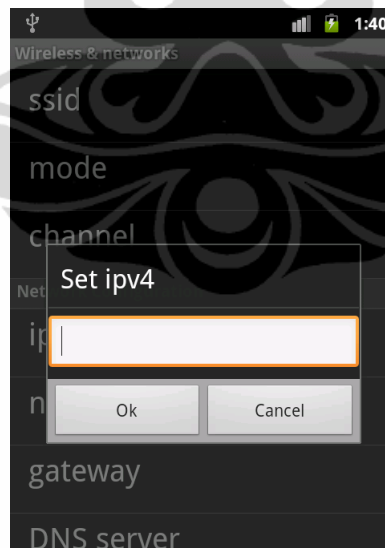


Figure 15 : Typical input dialog uses in network configuration section



### 3.5.3 Profile management

For easier configuration of Wi-Fi device, we made a simple profile management for Wi-Fi configuration. Basically, this profile management's task is saving the Wi-Fi parameters that showed on the Activity to a configuration file, then load it whenever user need it. Currently it provides these method:

save profile	Save a profile base on its SSID
delete profile	Delete profile saved on the system
load profile	Load a saved profile to the current configuration
apply profile	Apply the wireless & network settings

Table 2 : profile management function in Saje2Droid

To access this function, the user should click Android menu button, that will show an application's menu. The figures below show some of profile management features:

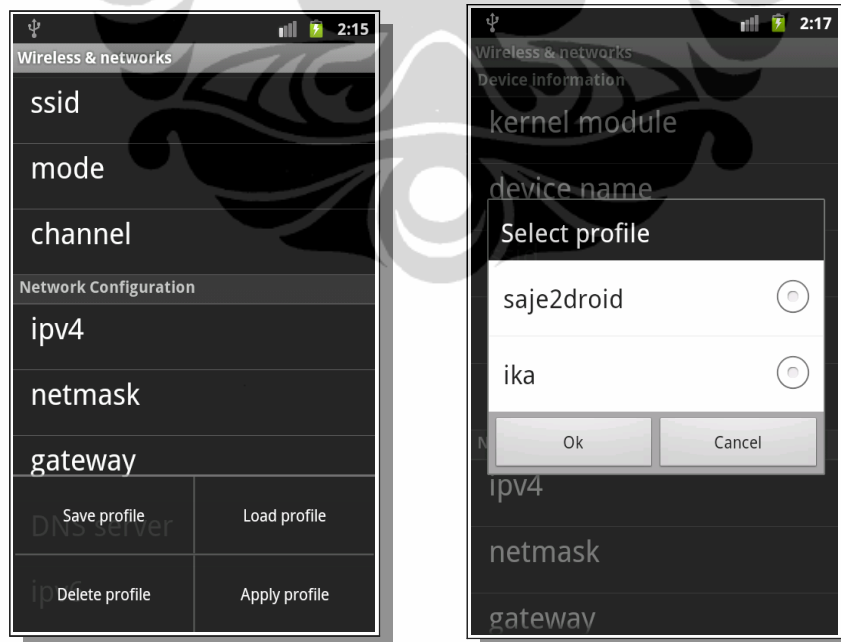


Figure 16 : Profile management menus

### 3.5.4 Connection information

The latest Android activity that we have developed in the Saje2Droid project was WifiRunActivity. This activity provides some basic info on an ad-hoc network of the Android device by showing current IPv4 address and SSID of the network. This activity will be launched when all of the profile configuration is applied successfully. It will show first a notification on the Android status bar. This notification is used to make sure that the WifiRunActivity is always running even the user start another application on Android. To test a connection between Android devices, we provides simple client-server application that can send/receive a message trough port 4444 UDP protocol The figure below show the WifiRunActivity:

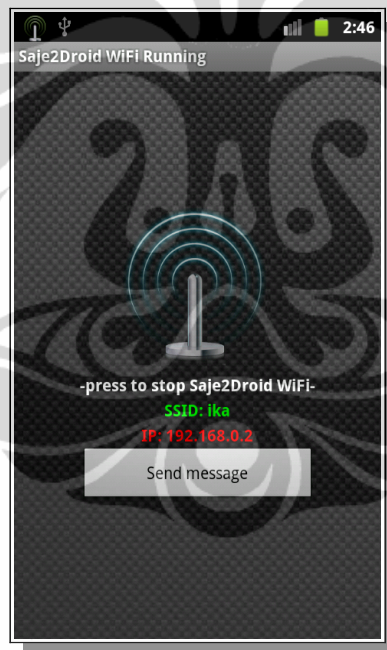


Figure 17 : WifiRunActivity

#### 4. CONCLUSION

Although there are some problems in the process of porting SAJE project to Android, with some adaptation, most of the code of SAJE project can be ported to Android, because Android is a Linux-based operating system. There are some parts that still need to be developed. SAJE currently only supports APM (Advanced Power Management) for battery management, for the future work it will be interesting to add the JNI code on a package `casa.saje.battery` to manage battery on Android operating system .

The current application user interface is very simple and it still needs an improvement. For example by adding a CPU and Memory usage visualization. And also it would be interesting to implement wireless network cell detection in the `WifiActivity` for the future work since SAJE already have a native function to get a list of wireless network cell.

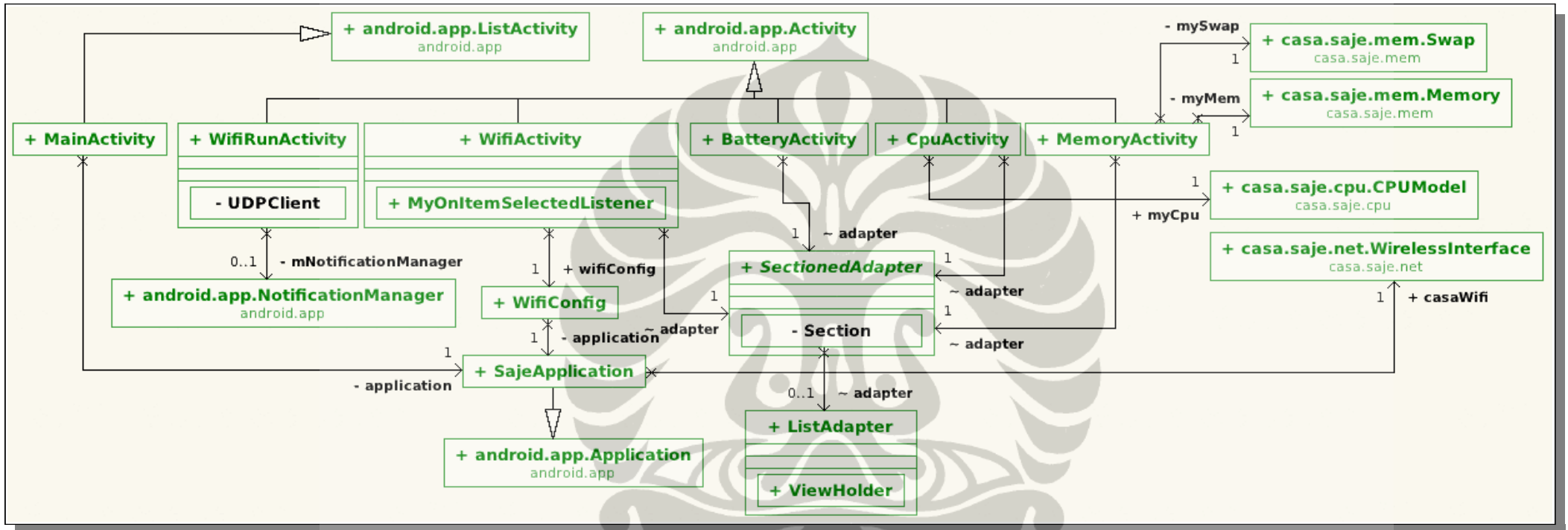
We have successfully set-up an ad-hoc communication mode in Android by using wireless tools. With this mode, we do not need to set up a network infrastructure to communicate Android device. In this version of `Saje2Droid` we have not yet implement the wireless security. Simple encryption schemes, such as WEP, is supported by wireless tools. This encryption scheme is usually easy to set-up. But for higher level of security, we would advise to use WPA encryption which have better key management and using AES instead of RC4.

## BIBLIOGRAPHY

- [1] **Pencil**, *Pencil project homepage*. [online, June 2011]  
<http://pencil.evolus.vn/en-US/Home.aspx>
- [2] **Xulrunner**, *Xulrunner homepage*. [online, June 2011] -  
<https://developer.mozilla.org/en/XULRunner>
- [3] **android-ui-utils**, *Android UI utilities – Pencil extension*. [online, June 2011] - <http://code.google.com/p/android-ui-utils>
- [4] **wireless-tools**, *Wireless tools for Linux*. [online, June 2011] -  
[http://www.hpl.hp.com/personal/Jean\\_Tourrilhes/Linux/Tools.html](http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html)
- [5] **net-tools**, *net-tools project*. [online, June 2011] -  
<https://developer.berlios.de/projects/net-tools/>
- [6] **Android market**, *BusyBox for Android*. [online, June 2011] -  
<https://market.android.com/details?id=stericson.busybox>
- [7] **Android market**, *SuperUser application for Android*. [online, June 2011]  
 - <https://market.android.com/details?id=com.noshufou.android.su&hl=en>
- [8] **Android documentation**, *Providing Resources in Android application*.  
 [online, June 2011] -  
<http://developer.android.com/guide/topics/resources/providing-resources.html>
- [9] **Android documentation**, *Android emulator*. [online, June 2011] -  
<http://developer.android.com/guide/developing/devices/emulator.html>
- [10] **wireless-tools**, *wireless-tools documentation*. [online, June 2011] -  
[http://www.hpl.hp.com/personal/Jean\\_Tourrilhes/Linux/Linux.Wireless.Extensions.html](http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html)



Appendix 1 : Class diagram of Saje2Droid project



## Appendix 2 : Class definition

## 1. MainActivity

<b>MainActivity</b>
- application: SajeApplication + MSG_TAG: String - menus: String[] [0..*]
+ onCreate() - showWarningMessage()

## 3. CpuActivity

<b>CpuActivity</b>
- adapter: SectionedAdapter - list1: String[] [0..*] + MSG_TAG: String + myArch: String + myBogo: String + myCpu: CPUModel + myFeat: String + myHard: String + myProc: String
+ onCreate()

## 4. MemoryActivity

<b>MemoryActivity</b>
- adapter: SectionedAdapter + MSG_TAG: String - memTitleList: String[] [0..*] - myMem: Memory + myMemFree: String + myMemSize: String + myMemUsed: String - mySwap: Swap + mySwapFree: String + mySwapSize: String + mySwapUsed: String - swapTitleList: String[] [0..*]
+ onCreate()

## 2. SajeApplication

<b>SajeApplication</b>
+ CASA_DATA_BIN_PATH: String + CASA_DATA_PATH: String + CONFIG_NET_RADIO: String + casaWifi: WirelessInterface - envPath: String - errMsg: String - kernMod: String + MSG_TAG: String + pd: ProgressDialog - result: int + scanner: Scanner + TMP_LSMOD1: String + TMP_LSMOD2: String - wifiDev: String + wifiEnabledAlready: boolean - wifiIsUp: boolean + wifiManager: WifiManager + wifiModLoaded: boolean
+ SajeApplication() - chmod() + cleanUp() - copyFile() - copyFile() + disableWifi() + displayToastMessage() + enableWifi() + getKernelMod() + getWifiConfList() - hasRootPermission() + initCheck() + installFiles() + killWpa_supplicant() + loadWifiModule() + mkdir() + rm() + runAsRootOnAndroid() + setNetParams() + setRadioOn() + setWifiParams() - setWifiPower() + unloadWifiModule()

## 5. WifiActivity

WifiActivity		
<ul style="list-style-type: none"> <li>- adapter: SectionedAdapter</li> <li>+ application: SajeApplication</li> <li>+ CASA_DATA_PATH: String</li> <li>+ channel: String</li> <li>+ devTitleList: String[] [0..*]</li> <li>+ dns_server: String</li> <li>+ essid: String</li> <li>+ gateway: String</li> <li>- hRefresh: Handler</li> <li>+ ipv4: String</li> <li>+ ipv6: String</li> <li>+ kernelMod: String</li> <li>+ lv: ListView</li> <li>+ MSG_TAG: String</li> <li>+ mode: String</li> <li>+ msg: String</li> <li>+ netTitleList: String[] [0..*]</li> <li>+ netmask: String</li> <li>- nid: NetworkInterfaceDiscovery</li> <li>+ pd: ProgressDialog</li> <li>- REFRESH: int</li> <li>+ result: String</li> <li>- selChannel: String</li> <li>- selConfigName: String</li> <li>- selMethod: int</li> <li>- selMode: String</li> <li>+ wifiConfig: WifiConfig</li> <li>+ wifiDev: String</li> </ul>		
<ul style="list-style-type: none"> <li>+ applySettings()</li> <li>+ deleteSetting()</li> <li>+ drawUI()</li> <li>+ inputKernelMod()</li> <li>+ inputWifiName()</li> <li>+ loadSetting()</li> <li>+ onCreate()</li> <li>+ onCreateOptionsMenu()</li> <li>+ onOptionsItemSelected()</li> <li>+ saveSetting()</li> <li>+ setChannel()</li> <li>+ setDns()</li> <li>+ setEssid()</li> <li>+ setGateway()</li> <li>+ setIPv4()</li> <li>+ setIPv6()</li> <li>+ setKernelMod()</li> <li>+ setMode()</li> <li>+ setNetmask()</li> <li>+ setWifiName()</li> </ul>		
<table border="1"> <thead> <tr> <th>MyOnItemSelectedListener</th> </tr> </thead> <tbody> <tr> <td> <ul style="list-style-type: none"> <li>+ onNothingSelected()</li> <li>+ onItemSelected()</li> </ul> </td> </tr> </tbody> </table>	MyOnItemSelectedListener	<ul style="list-style-type: none"> <li>+ onNothingSelected()</li> <li>+ onItemSelected()</li> </ul>
MyOnItemSelectedListener		
<ul style="list-style-type: none"> <li>+ onNothingSelected()</li> <li>+ onItemSelected()</li> </ul>		

## 6. WifiConfig

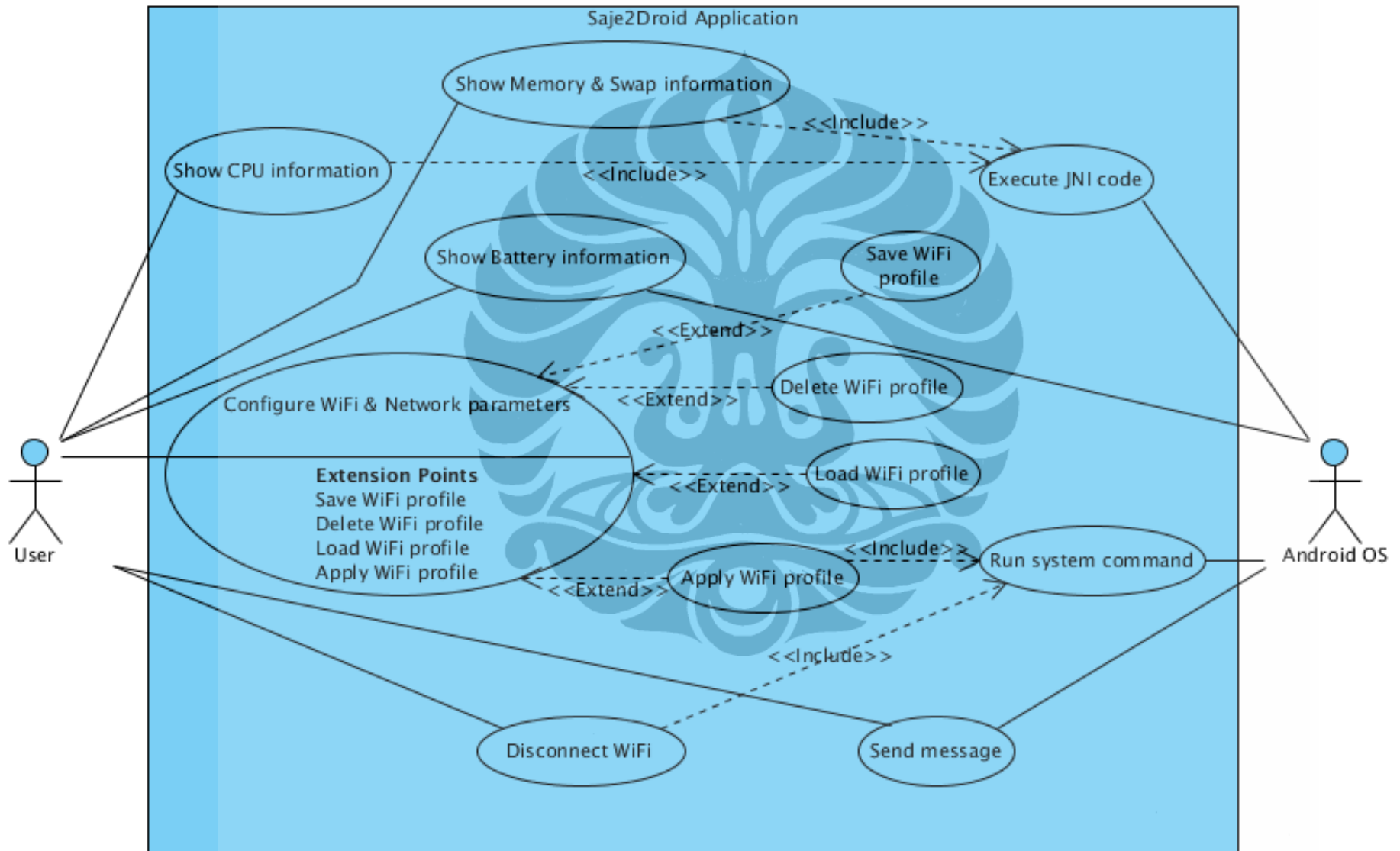
WifiConfig
<ul style="list-style-type: none"> <li>- application: SajeApplication</li> <li>- CASA_DATA_PATH: String</li> <li>+ configName: String</li> <li>+ filename: String</li> <li>- serialVersionUID: long</li> <li>- WIFI_CONF_DIR: String</li> </ul>
<ul style="list-style-type: none"> <li>+ WifiConfig()</li> <li>+ delete()</li> <li>+ read()</li> <li>+ readLinesFromFile()</li> <li>+ write()</li> <li>+ writeLinesToFile()</li> </ul>

## 7. WifiRunActivity

WifiRunActivity			
<ul style="list-style-type: none"> <li>- animation: Animation</li> <li>- btnBroadcast: Button</li> <li>- client: Thread</li> <li>+ done: boolean</li> <li>- hNotify: Handler</li> <li>- ip: TextView</li> <li>- mNotificationManager: NotificationManager</li> <li>- MSG_TAG: String</li> <li>- msg: String</li> <li>- NOTIFICATION_ID: int</li> <li>- NOTIFY: int</li> <li>- notif: Notification</li> <li>- sender: String</li> <li>- server: Thread</li> <li>+ serverAddr: InetAddress</li> <li>+ serverport: int</li> <li>+ socket: DatagramSocket</li> <li>- ssid: TextView</li> <li>- wifiRun: ImageView</li> </ul>			
<ul style="list-style-type: none"> <li>+ WifiRunActivity()</li> <li>+ callNotify()</li> <li>+ onCreate()</li> <li>+ onDestroy()</li> <li>+ sendMessage()</li> <li>+ showMessage()</li> </ul>			
<table border="1"> <thead> <tr> <th>UDPCClient</th> </tr> </thead> <tbody> <tr> <td> <ul style="list-style-type: none"> <li>- ip_address: String</li> <li>- message: String</li> <li>+ serverport: int</li> </ul> </td> </tr> <tr> <td> <ul style="list-style-type: none"> <li>+ UDPCClient()</li> <li>+ run()</li> </ul> </td> </tr> </tbody> </table>	UDPCClient	<ul style="list-style-type: none"> <li>- ip_address: String</li> <li>- message: String</li> <li>+ serverport: int</li> </ul>	<ul style="list-style-type: none"> <li>+ UDPCClient()</li> <li>+ run()</li> </ul>
UDPCClient			
<ul style="list-style-type: none"> <li>- ip_address: String</li> <li>- message: String</li> <li>+ serverport: int</li> </ul>			
<ul style="list-style-type: none"> <li>+ UDPCClient()</li> <li>+ run()</li> </ul>			



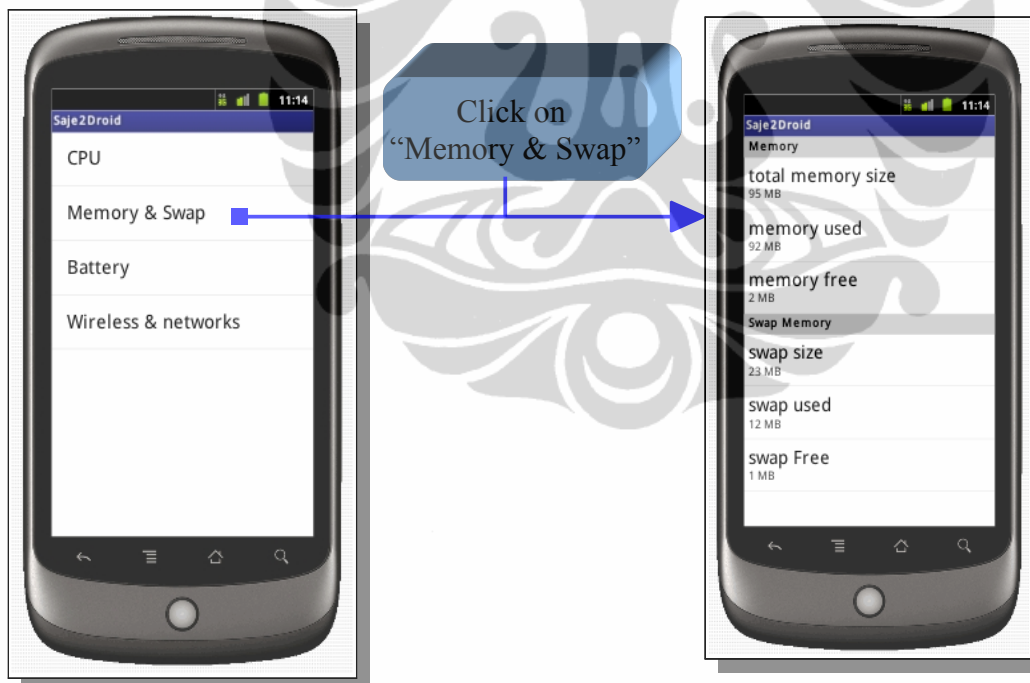
Appendix 3 : Use case diagram



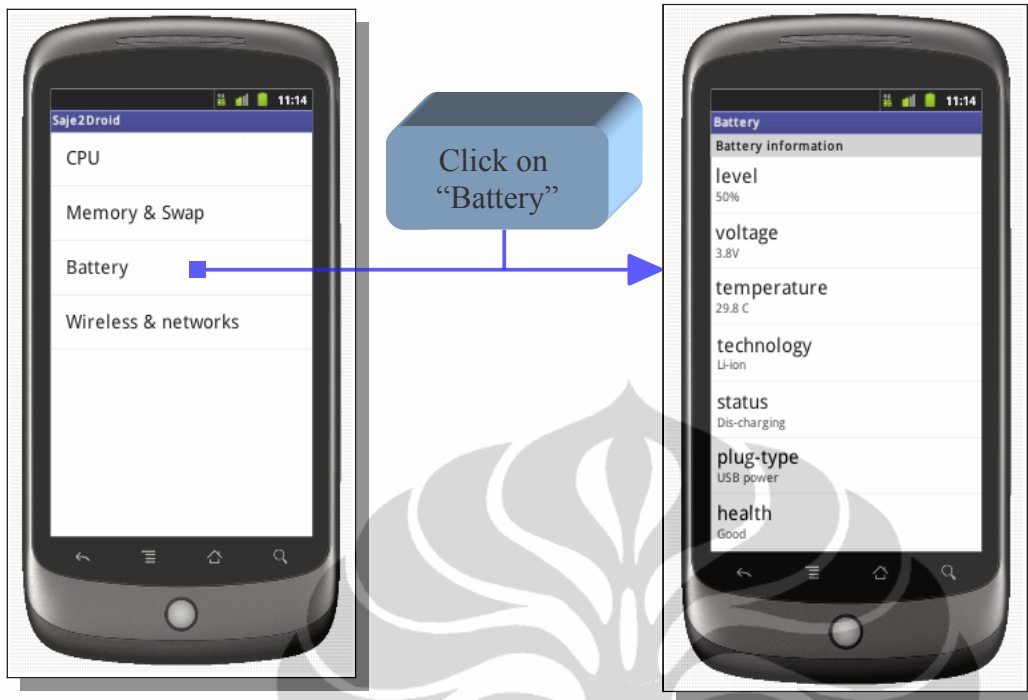
## 1. Show CPU information



## 2. Show Memory information



### 3. Show Battery information



### 4. Configure WiFi & network

