Development of a tool for analysis and visualization of Android logs

Fadwa Rebhi

Ref : 2013/II3 *presented on September 2013*

University of la Manouba

National School of Computer Sciences

Graduation Projet Report

Performed in order to obtain

National Diploma of Engineering in Computer Sciences

By :

Fadwa REBHI

Subject matter :

Development of a tool for analysis and visualization of Android logs

Hosting company : INRIA, Nancy, France

Monitored by : M. Abdelkader LAHMADI

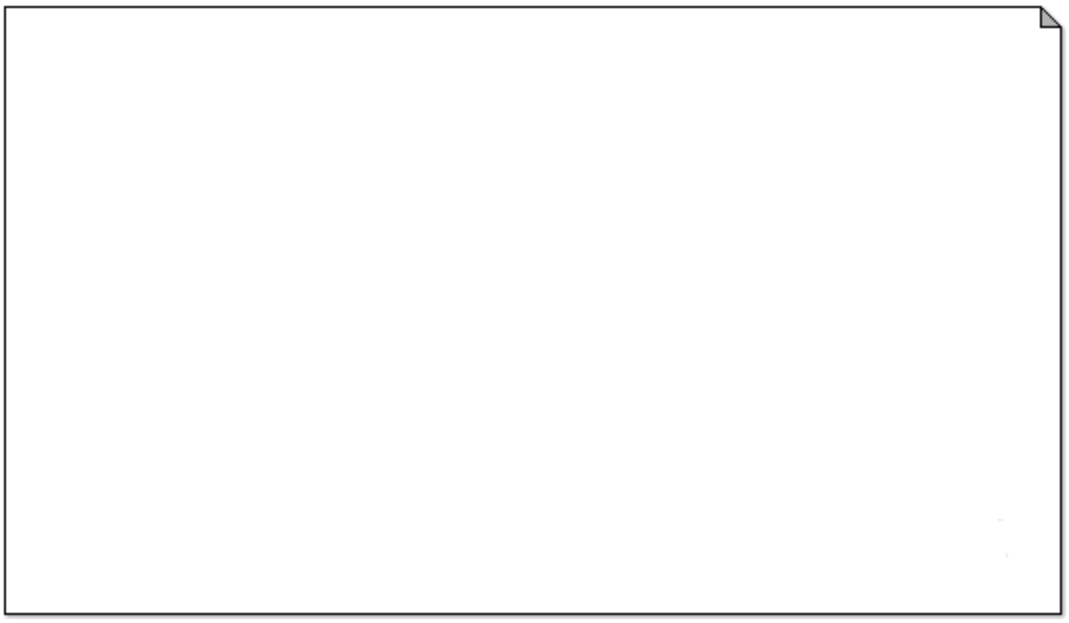
Supervised by : Mme Hanen IDOUDI

Adress : 615, street Jardin Botanique, 54600, VILLERS LES NANCY, Nancy , France

Phone : +33 (0)3 83 59 30 00

Signature of the monitor

Mr. Abdelkader LAHMADI



Acknowledgement

It is not possible to prepare a project report without the assistance and encouragement of other people. This one is certainly no exception.

On the very outset of this report, I would like to extend my sincere and heartfelt obligation towards all the personages who have helped me in this endeavor. Without their active guidance, help, cooperation and encouragement, I would not have made headway in the project.

I would like to express my profound gratitude and deep regards to M.Abdelkader LAHMADI, my supervisor at LORIA laboratory, for his exemplary guidance, monitoring and constant encouragement throughout the course of this project.

I would like to thank Mme. Hanen IDOUDI, my internship supervisor, for meticulously reviewing and validating this work and checking for its technical correctness. She deserves special praise for his availability, attention to detail and valuable insight.

Many thanks go to Zied FERCHICHI, a computer engineer for his continuous encouragement all throughout this internship and also for his critical comments and beneficial suggestions.

Last but not least, I would like to express my respect and gratitude to the jury members for the honor they have granted me to have wished to examine and evaluate this modest contribution, and all of my professors at the National School of Computer Sciences for the schooling and training they have given me and hopefully that this work will meet up to their standards.

Abstract

Since they are open, programmable, networked devices, smartphones have become susceptible to various threats. Unfortunately, malicious applications are increasing rapidly and are causing serious damages for users and for their privacy. Thus, it is important to study and know what applications are really doing on Android devices.The purpose of this project is to develop a tool that can display behaviors graphs for Android applications. This will be accomplished by analyzing and visualizing Android Logs, using HBase system and treemapping technique.

Keywords : Android applications, behaviors, logs, HBase, treemapping.

Résumé

Comme ils sont programmables, ouverts, interactifs en réseau, les smartphones sont devenus sensibles à diverses menaces. Malheureusement, les applications malveillantes se multiplient rapidement et causent de graves dommages pour les utilisateurs et pour leur vie privée. Ainsi, il est important d'étudier et de savoir le comportement réel des applications au sein des équipements Android. Le but de ce projet est de développer un outil qui permet d'afficher des graphes de comportements pour les applications Android. Cet objectif sera atteint par l'analyse et la visualisation des journaux d'événements Android tout en utilisant le système HBase et la technique de treemapping.

Mots clés : Android applications, comportements, journaux d'événements, Hbase, treemapping.

Contents

General introduction **1**

General context **3**

1 Preliminary study **7**

1.1 State of the art 7

1.1.1 Android environment 7

1.1.2 Mobile malware 8

1.1.3 Android application components 8

1.1.4 Android logging system 10

1.1.4.1 Logcat 11

1.1.4.2 Dumpsys 12

1.2 Study of existing solutions 15

1.2.1 Androguard 15

1.2.2 DroidBox 16

1.3 Project approach 17

1.4 Conclusion 18

2 Analysis and specification of requirements **19**

2.1 Requirements analysis 19

2.1.1 Functional requirements 19

2.1.2 Non functional requirements 21

2.2 Use cases 22

2.2.1 General use case 22

2.2.2 Detailed use case 24
2.3 Sequence diagrams : 27
2.3.1 Scenario of application visualization by a simple user : 27
2.3.2 Scenario of application visualization by an analyst 28
2.4 Conclusion 30

3 Design and structure **31**
3.1 Global design 31
3.2 Detailed design 33
3.2.1 System architecture 33
3.2.2 MVC layers designs 33
3.2.2.1 Data layer design 33
3.2.2.2 Controller layer design 35
3.2.2.3 View layer design 43
3.3 Conclusion 44

4 Implementation details **45**
4.1 Work environment 45
4.1.1 Hardware environment 45
4.1.2 Software environment 46
4.2 Development process 47
4.2.1 Logs analysis 47
4.2.2 Logs visualization 49
4.2.3 Graphs interpretation 50
4.3 Tool screenshots 53
4.4 Conclusion 64

Bibliography **65**

Netography **66**

Appendix **68**

List of figures

1.1 Android application components types [N5] 9
1.2 Androguard output [N10] 16
1.3 DroidBox report [N11] 17

2.1 General use case 23
2.2 Send information use case 24
2.3 Visualize the behavior of a chosen application use case 25
2.4 Consult the list of applications of a chosen device use case 26
2.5 Manage analysts use case 26
2.6 Application visualization by a simple user 28
2.7 Application visualization by an analyst 30

3.1 MVC pattern 32
3.2 System architecture 33
3.3 Collimator probe functionalities 34
3.4 Package diagram of the controller 36
3.5 Classes diagram of the controller 38
3.6 Logger class diagram 39
3.7 Dumper class diagram 40
3.8 XmlGenerator class diagram 41
3.9 ToolDealer class diagram 42
3.10 View class diagram 43

4.1 Paintball graph 51

4.2 Facebook graph 51
4.3 Viber graph 52
4.4 Telephony graph 53
4.5 Home page 54
4.6 Analyst home page 54
4.7 Analyst registration interface 55
4.8 Devices view 56
4.9 Applications view 56
4.10 Application behavior graph 57
4.11 Logcat information interface 58
4.12 Logcat attributes view 58
4.13 Dumpsys information view 59
4.14 Dumpsys attributes view 60
4.15 Zooming view 60
4.16 Administrator interface 61
4.17 Management view 62
4.18 Simple user homepage 62
4.19 Simple user applications view 63
4.20 Simple user application view 64

List of tables

3.1 Logs table 35

General introduction

Throughout recent years, Android applications have been evolving in a remarkable way because android phones are widespread and gaining popularity. Since these applications have various functions, they acquire various permissions and thus become integrated with our lives, being able to access our email, social networking accounts, financial information, personal photos and even our cars and homes.

Because of the accessibility of Android market, anyone can develop an Android application and upload the developed application on the android market. Since they are open, programmable, networked devices, smartphones have become susceptible to various threats. Unfortunately, malicious applications are increasing rapidly. They cause serious damages, such as making the phone unusable, stealing private information, and sending SMS or phone call for unwanted billing to users. Therefore, it is important to study and analyze efficiently what mobile applications are really doing on Android devices.

Our graduation project fits within this context. We will focus on discovering the behavior of Android applications. In fact, we envision to design a generic and modular tool generating execution models of Android applications. The purpose of our project is to establish behaviors graphs for android applications which represent the different components with detailed information describing every single event related to their executions. This will be accomplished by taking advantage of powerful Android tools which record everything about what happens in Android devices.

The structure of our report will be as follows : the first chapter entitled « Preliminary study », introduces the main terminology related to our project and presents a study of the existing similar products and our added value. The second chapter named « Requirements Analysis and Specification» specifies the functional and non-functional requirements followed by the use case models and scenarios examples of the system. The third chapter entitled « Architecture and Design of the Solution », gives an overview of the architecture of the proposed solution and describes the general and detailed design. Finally, the last chapter « Implementation details » describes the development environment and provides an overview of the achieved work.

General context

Throughout this part, we put this project into its general framework. We focus primarily on presenting the host organization and laboratory, then presenting the host team and finally, exposing the work goals.

Presentation of the host organization : INRIA [N1]

Public science and technology institution established in 1967, Inria is is the only public research body fully dedicated to computational sciences. Combining computer sciences with mathematics, Inria's 3,400 researchers strive to invent the digital technologies of the future.

Educated at leading international universities, they creatively integrate basic research with applied research and dedicate themselves to solving real problems, collaborating with the main players in public and private research in France and abroad and transferring the fruits of their work to innovative companies. The researchers at Inria published over 4,800 articles in 2010. They are behind over 270 active patents and 105 start-ups. In 2010, Inria's budget came to 252.5 million euros, 26% of which represented its own resources.

Presentation of the host laboratory : LORIA [N2]

LORIA is the French acronym for the "Lorraine Research Laboratory in Computer Science and its Applications". It a research unit created officially in 1997 and which is common to CNRS, the University of Lorraine and INRIA.

The laboratory is a member of the Charles Hermite Federation, which groups the four main research labs in Mathematics, in Information and Communication Sciences and in Control and Automation.

The laboratory is structured in five departments :

- Algorithms, Computation, Image and Geometry Department : It connects six teams, sharing scientific interests in some structures, the algorithms to construct or detect them, and their mathematical or algorithmic properties. Such structures typically play a central role in geometric and imaging problems. The department studies several aspects of these structures (combinatorics, algebra, numerics ...) and applies them in different domains (bioinformatics, modeling ...).

- Formal Methods department : It loosely shares common concepts, techniques and tools. It develops the research topics like contribution to logics and proof theory, techniques for the verification of distributed and reactive systems, virology and safety.

- Networks, Systems and Services : The teams of this department tackle problems related to large parallel and distributed systems and the ability that they offer to work, manage, compute and communicate seamlessly on a previously unknown scale, accuracy and spacial distribution. Particular emphasis lies on management and cooperation in modern networks, parallel and distributed computing, and real time aspects. The palette of the applied methods comprises formal modeling and proving, design of algorithms and protocols, implementation of software, experimenting, benchmarking, and the provisioning of research platforms for emulation, simulation and real scale systems.

- Knowledge and language management department : It deals with the three closely related notions of document, language and knowledge. In a wide range of contexts, computational semantics (studying, building and reasoning with meaning) is a common research area. Many activities provide input or output to an end-user, for example in developing natural language-based interface, learning systems, or providing ontologies to experts of a domain.

- Complex Systems and Artificial Intelligence : The scientific activities of this department cover a rather wide range of topics. It's primarily interested in the computational aspects usually related to the notion of complex systems defined as systems of many entities that simultaneously and locally interact closely, and for which the study of the evolution is out of reach of a pure mathematical approach.

Presentation of the team : MADYNES

The research team MADYNES is one of the teams of Networks, Systems and Services department in LORIA. It deals with the design, validation and implementation of new supervision and control paradigms and architectures capable of controlling the increasing dynamicity of services and resistant to scalability induced by the ubiquitous Internet.

The activities of this team cover 4 basic topics :

- Vulnerabilities Management
- Large-scale monitoring
- Network management
- Co-simulation of dynamic ambient networks

Work goals

Our work is a part of Android Environments security (AES). AES deals primarily with vulnerabilities and malicious applications of android devices. In order to facilitate this detection, we aim through this project to design a tool that can describe clearly the behavior of android applications. This tool is supposed to provide users with understandable information regarding what mobile applications are really doing on android devices.

Conclusion

In this part, we put this project in its general context by presenting the host institution, the host team and work goals. In the next chapter, we will focus on the state of art of our project.

Chapter 1

Preliminary study

In the following chapter, we present the state of the art of our project where we treat the different concepts we have studied in order to understand the project. Then, we expose some tools dealing with android application analysis. Finally, we expose our approach.

1.1 State of the art

In this section, we treat diverse concepts. We begin by presenting the Android environment. Then, we expose the different components that construct android applications. After that, we detail the core basics of the Android logging system.

1.1.1 Android environment

Android is a software platform for mobile devices that includes an operating system, middleware and key applications. It is based on the Linux operating system and developed by Google and the Open Handset Alliance.
It was built from the ground-up to enable developers to create compelling mobile applications that take full advantage of all a handset has to offer. It was built to be truly open. For example, an application can call upon any of the phone's core functionality such as making calls, sending text messages, or using the camera, allowing developers

to create richer and more cohesive experiences for users.
Applications are written using the Java programming language and run on Dalvik, a custom virtual machine specifically designed for embedded use. Android relies on modified Linux Kernel version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. It also includes a set of Java libraries that provide the functionalities available in standard Java Programming Language, and C/C++ libraries such as SQLite relational database management system, 3D libraries, Media Libraries etc. [N3]

1.1.2 Mobile malware

Malicious software ("malware") that is designed specifically to target a mobile device system, such as a tablet or smartphone to damage or disrupt the device. It is broadly categorized to include viruses, botnets, worms, and Trojan horses [N4]. Most mobile malware is designed to disable a mobile device, allow a malicious user to remotely control the device or to steal personal information stored on the device. Mobile malware can do just about everything a virus on a PC can do and more. Viruses can bog down the phone's processing speeds, hack the email of the user and send out spam to all his contacts, randomly delete his important files and calendar entries, and initiate system-wide crashes. One of the worst and most recent viruses that targets only mobile phones eventually causes the infected phone to lock up completely, making it unusable. However, some smartphone malware can access camera and photos. It can steal all the images saved on the camera and actually hijack the camera to take photos at random.

1.1.3 Android application components

Application components are the essential building blocks of an Android application. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. There are mainly five types of components by which an Android Application is made of.

These components are illustrated by figure 1.1 :

FIGURE 1.1 – Android application components types [N5]

- **Activities :** Activity is an individual user interface screen in an Android application where visual elements called Views can be placed and the user can perform various actions by interacting with it.
- **Services :** A service is an Android application component that run in background and has no visual user interface. Services are used to perform the processing parts of your application in the background. While the user is working on the foreground user interface services can be used to handle the processes that need to be done in the background.

  A service can be started by another Android application components such as an activity or other services and it will continue to run in the background even after the user switches to another application.
- **Content providers :** Content providers in Android provides a flexible way to make data available across applications. This is the mechanism used to expose many of a device's data resources for retrieval and update : Contacts, media store, bookmarks, phone-call log, and so on.

- **Broadcast receivers :** Broadcast receivers are used to receive messages that are broadcasted by the Android system or other Android applications. For example, an application can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- **Intents :** Intents are the component activating mechanism in Android. They constitute the core message system in Android and define messages to activate a particular component. The most common use of intents is to bind your application components. Intents are used to start, stop and transit between the activities within an application. [B1]

1.1.4 Android logging system

Event logs are text files that record chronologically the events executed by a server or computer application. Each action of a computer system (registration, program installation, web browsing ...) produces a log file. An effective logging strategy has long been recognized by system administrators as a way to keep track of problems with components and applications, provide quantifiable statistics for an application's history, help in troubleshooting issues, as well as help in monitoring the overall health of systems. [N6]

Android is based on a Logging system which keeps a record of the execution of various applications. These logs contain a lot of information about what is happening on the phone (start/stop an application, access to shared data, connection status change, etc).

Android provides 2 different tools allowing to get access to these logs : logcat and dumpsys.

1.1.4.1 Logcat

Logcat is a standalone program for viewing log messages contained in logs buffers. These logs contain a number of metadata fields. Every Android log has a timestamp, an UID(User IDentifier), a PID(Process IDentifier), a priority and a content. The timestamp refers to the date, and invocation time of the log. The UID and PID refer respectively to the application and the process issuing the message. The priority refers to the degree of importance of the log. It is one of the following character values, ordered from lowest to highest priority : V (Verbose), D(Debug), I(Info), W(Warning), E(Error), or F(Fatal). The content represents details of the message which include precise information and have not a standard format. [N7]

Logcat messages can be used to identify the life cycle of an Android application and recognize its behavior. In fact, there are logcat messages which keep record of all events related to an application's components.

Here some examples of these logs :

Intent creation :

03-29 09 :27 :58.240 280 2871 I am_new_intent :
[10866059362,com.sec.android.app.twlauncher/.Launcher,android.intent.action.MAIN,
NULL,NULL,274726912]

Service creation :

03-29 04 :47 :32.559 2806 2869 I am_create_service :
[1085277064,com.facebook.katana/.service.FacebookService,act=com.facebook.katana.service.51,
14447]

Activity restart :

03-29 06 :55 :00.005 2806 2812 I am_restart_activity :
[1085169744,10,com.sec.android.app.clockpackage/.alarm.AlarmSmartAlert]

Content provider update :

03-28 21 :47 :11.140 14447 18610 I content_update_sample :
[content ://com.facebook.katana.provider.CacheProvider/cache/name/notifications_history,
update,201,41]

Broadcast receiver start :

03-29 12 :12 :13.175 2806 2981 I am_proc_start :
[28450,10063,com.google.android.apps.uploader,broadcast,
com.google.android.apps.uploader/.ConnectivityBroadcastReceiver]

1.1.4.2 Dumpsys

Dumpsys is a tool to gather various information from Android device. It is a very interesting android shell command that dumps the state of various running services and also other system information. [N8]

In fact, it allows to save a snapshot of the system memory to a dump file. There are times when system memory requires analysis but it may not be possible to halt the system and take a normal crash dump. Many problems can be resolved by taking a snapshot of the system memory while the system is running. The dumpsys command performs this function after it determines that there is enough file system space to save a core dump. [N9]

After executing the command, there are various service messages are outputted. The first part lists the available services for dumping. In the list, various functional services running on Android device are mentioned like location, connectivity, notification, meminfo and activity. Then, a detailed dump of each service is listed in a simple string representation.

Keeping track of intents, services, activities, broadcast receivers and content providers of an application is efficient in identifying its whole behavior. Fortunately, Android's dumpsys command can show a snapshot of the current state of affairs. Typing «adb shell dumpsys activity» in a development computer's command window outputs an up-to-date list of actions and events related to all application components.

Information found in dumpsys files differ from those found in logs messages. Thus, we can collect additional details about the different components of an Android application. In fact, we find a block dealing with activities where we can find a list of activities records containing numerous characteristics about the activity name, the package implementing it, the location of its code source, device configuration information that can impact the resources used by the current activity and other parameters. Here an example of an activity record belonging to Facebook application.

```
ActivityRecord427d0088 com.facebook.katana/.activity.FbFragmentChromeActivity
packageName=com.facebook.katana processName=com.facebook.katana frontOfTask=true
realActivity=com.facebook.katana/.activity.FbFragmentChromeActivity
base=/data/app/com.facebook.katana-1.apk
data=/data/data/com.facebook.katana
stopped=true visible=false sleeping=true fullscreen=true immersive=false
```

The activity is entitled «FbFragmentChromeActivity» and located in /data/data/com.facebook.katana. The visible attribute is set true, thus, this activity is needed to be shown. Sleeping, when set to true, means that the activity has been told to sleep. The immersive mode is not active ; thus this activity cannot be interrupted with other activities or notifications. The activity covers the full screen and it is waiting for another activity to be visible.
In the dumpsys file, we find a list of active services records during command execution. Each

service record contains information about the name, the process, the application, the location and other details defining the current service. The following block describes a service record.

```
ServiceRecord4251aa68 com.google.android.youtube/.core.transfer.DownloadService
packageName=com.google.android.youtube processName=com.google.android.youtube
baseDir=/data/app/com.google.android.youtube-1.apk
dataDir=/data/data/com.google.android.youtube
createTime=-19m0s343ms executingStart=-18m59s300ms restartTime=19m0s342ms
```

The service is named DownloadService. It is part of Youtube application presented by the package« com.google.android.youtube». The baseDir and dataDir attributes reprensent the lo-

cation repositories of respectively the installation file and application resources. In the end, the creation, the executing and the restart times are mentioned.

Dumpsys keeps track of intents which represent the component activating mechanism in Android. The following example is a record of a start service intent used by Facebook application to start a service entitled belonging to the same application :

```
PendingIntentRecord41965290 com.facebook.katana startService uid=10236
packageName=com.facebook.katana type=startService flags=0x0
requestIntent=cmp=xxxx
```

All messages broadcasted by the Android applications are also recorded. We mention in the following block an example of a registered broadcast receiver used by the android telephony application. The PID and UID attributes refers to the process and application identifiers dealing with it. This broadcast receiver is associated to a filter which specifies which intents is listening for. In our example, it is listening to the native ACTION_BOOT_COMPLETED which fires once the device has completed its start-up sequence.

```
ReceiverList{4186e4e0 474 com.android.phone/1001 remote :417084a8}
pid=474 uid=1001 Filter 0 : BroadcastFilter{418698b0}
Action :"android.intent.action.BOOT_COMPLETED"
```

Contents providers related information are outputted by dumpsys command. A list of the recent content providers used by diverse applications executed in the android device is cited. Here an example of registered details about a content provider issued by Google maps application :

```
LayerInfoProvider(com.google.android.apps.maps/com.google.googlenav.layer.LayerInfoProvider)
package=com.google.android.apps.maps process=com.google.android.apps.maps
proc=ProcessRecord{423ec890 2935 :com.google.android.apps.maps/10061}
```

When debug engineering is necessary, a lot of useful information from dumpsys can be retrieved for debugging purpose. Thus, dumped information allows to check how an application behaves and how it affects the overall device.

1.2 Study of existing solutions

Many researches have been done in order to deal with android application analysis.In this section, we expose some of the available tools trying to figure out their features and depict their weaknesses.

1.2.1 Androguard

Androguard is mainly a tool written in python for android applications analysis. With its internal decompiler, it retrieves the Java source code of an application from the bytecode. By analyzing the code lines, useful details about the code, its basic blocks, its instructions and the permissions needed can be figured out.

Through an interactive shell, displaying a method, accessing to each field of an instruction, annotating a specific method or an instruction and even renaming classes are possible.
Figure 1.2 depicts the sort of information displayed by Androguard.

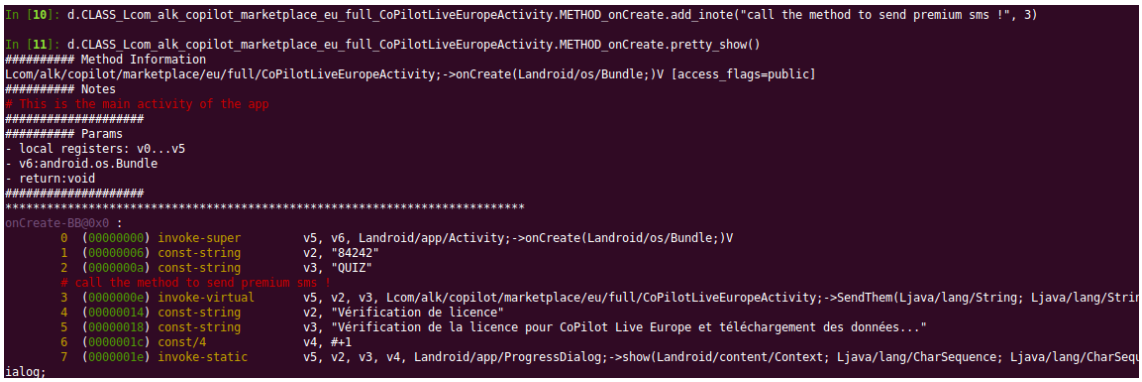


FIGURE 1.2 – Androguard output [N10]

As shown in the figure above, information outputted by Androguard are hardly understandable, especially for complex applications. In fact, information about source code will be displayed whithout giving logical relationship between its different portions. Thus, the identification of applications behaviors could not be an easy process.

1.2.2 DroidBox

DroidBox is a standbox which offers dynamic analysis of Android applications. It gives interesting clues about application behavior by providing network traffics, accessed files, data leakages, needed permissions, cryptography operations and SMS/Phone calls.

These details are visualized through a graphical report as shown in figure 1.3 :

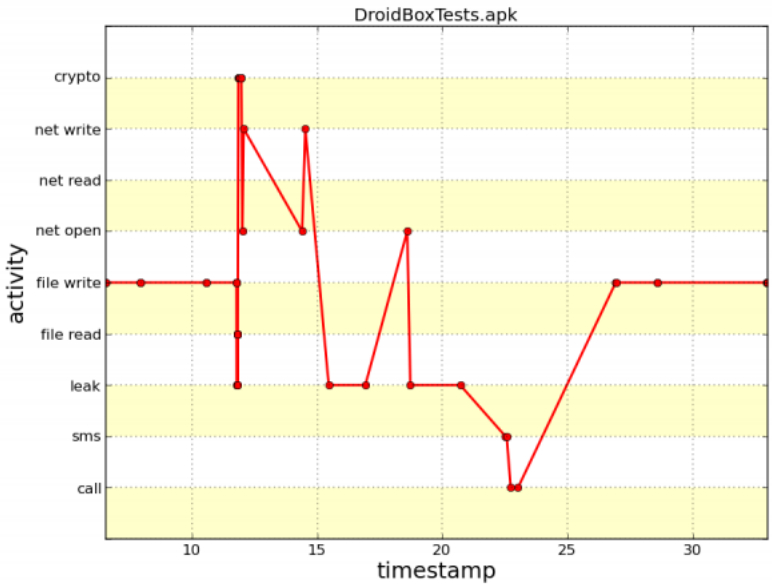


FIGURE 1.3 – DroidBox report [N11]

DroidBox represents some useful information dealing with the interaction between Android applications and the whole system. It does not give any information about applications components and their interrelationships.

1.3 Project approach

Without knowing the source code of applications, we will try to construct execution models of Android applications. The purpose of our project is to establish for android applications behavior graphs which represent the different components with detailed information describing every single event related to its executions. Thus, we aim to take advantage from Android logs which record everything about what happens in android devices.

Our envisioned tool is designed to android applications understanding of different users. By centralizing logs information in specific server, we obtain dynamic and up-to-date graphs. Observing applications behaviors in numerous users's devices may be efficient in detecting the normal and the malicious ones.

1.4 Conclusion

The preliminary study allowed us to understand the basic concepts of the project and to analyze the strengths and weaknesses of some of the existing Android applications analysis tools. Therefore, this allowed us to get a clearer idea about our envisioned tool and to further refine the whole system's requirements which will be detailed in the next chapter.

Chapter 2

Analysis and specification of requirements

After having provided the theoretical basics that are necessary for understanding the project's concepts, we now start the specification's phase. At first, we are going to list user's needs. Then more detailed specifications and analysis will be made through use cases diagrams. Finally, we will expose some sequence diagrams in order to bring out the behavioral aspect of the application.

2.1 Requirements analysis

In this section, we detail all functional requirements and non-functional ones that must be taken into consideration during our tool development process.

2.1.1 Functional requirements

Our envisioned tool is designed to analyze and visualize Android applications executed in different Android devices. In order to perform these basic functionalities, data from applications users devices are needed.

Thus, the whole system has to address different functional requirements for the analyst who will have a global view of all applications of all users and for the simple user who, after making data retrieved from his device available, will visualize only his applications.

So, we can define the following functional requirements depending on the type of system user :

- **Simple user :**

  Send logcat messages :

  The identification of an application behavior is strongly based on logcat messages which are internal to the Android device. Thus, the system should permit to user to make this sort of information available for use and exploration.

  Send dumpsys messages :

  The rich content of dumpsys files is efficient in the analysis process of Android applications. So, the user should have the possibility to provide this valuable data for analysts.

  Authenticate :

  In order to get access to the application, the simple user should be authenticated. Authentication is done through the use of Android device's identifier.

  Consult the list of applications :

  The simple user could consult the list of applications executed on his device and choose which application to visualize.

  Visualize applications behaviors related to his device :

  The system should allow the Android user to visualize the application's behavior and display detailed information about its different components.

- **Analyst :**

  Authenticate :

  In order to get access to the application, the analyst should authenticate. Authentication is done through the use of logon passwords (login+ password).

  Consult the list of devices involved in the analysis process :

  The system should allow the analyst to consult the list of the contributing devices and choose which device to explore.

  Consult the list of applications executed on each user device :

  The analyst should have the possibility to consult the list of applications available in the concerned device.

Visualize applications behaviors :

The analyst could visualize the application's behavior of each user and display detailed information about its basic components.

We can define also the administrator, who is an analyst, having the additional task of managing analysts. His core management functions are :

Adding analysts :

The administrator has the possibility to add analysts by confirming their registrations submissions.

Deleting analysts :

The administrator has the possibility to delete analysts.

Giving the administrative rights to analysts :

The system should allow the administrator to give analysts the right of administration.

2.1.2 Non functional requirements

The required system has to deliver specific and measurable improvements in :

Availability : The system must be continuously available.

Fluidity : As every posted data in the application needs to be updated periodically, refresh actions must be transparent for end users to maintain tool stability.

Response time : The user should find out the information needed as fast as possible so generated graphs have to be instantly and easily understood.

Usability : The tool should also be intuitive and simple to use. The analyst should be able to easily process the actions needed without any special training.

Maintainability : The different modules of the application must be legible and understandable in order to maintain and update them quickly and easily.

Security : The system should ensure a considerable level of security.Thus,the access to the application should be restricted via authentication mechanisms.

2.2 Use cases

To specify in a formal way the required needs by the application, we opted for the realization of some Use Cases Diagrams to have a better understanding of the needs.

2.2.1 General use case

The following use case diagram presents the general functionalities performed by the system. It mentions for each type of users the different services offered for him.

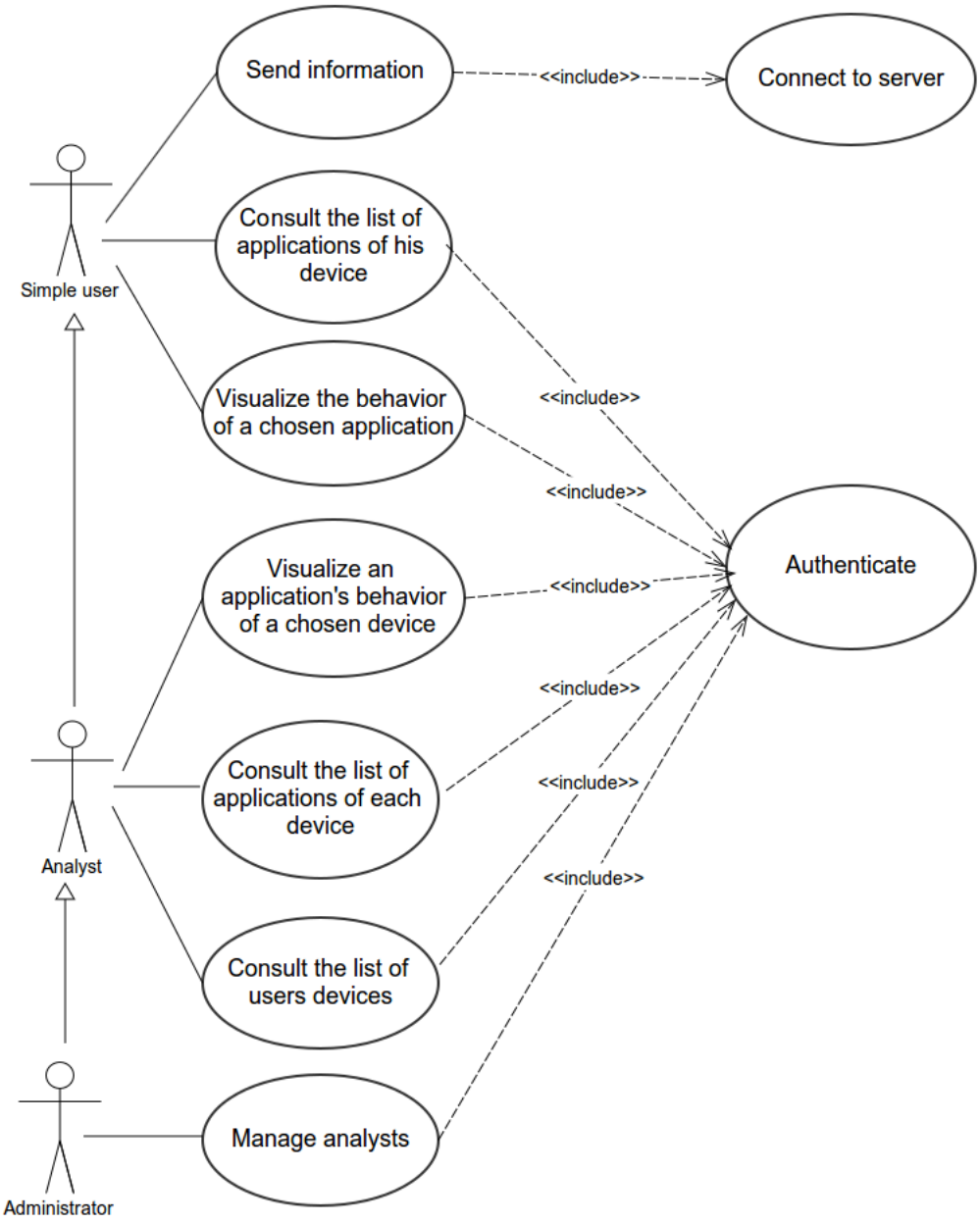


FIGURE 2.1 – General use case

2.2.2 Detailed use case

In this section, we focus on refining use cases to better clarify the requirements. We choose for each type of users some use cases to detail.

- **Simple user :**

  Send information : detailed use case

  Android logging system is a basic concept of our approach. Without the availability of logs, the analysis process would not be possible. End-users of mobile applications should contribute to achieve this purpose. Collecting logcat messages contained in logs buffers of each mobile device and then sending them to a server for storage must be performed. Dumpsys files contain a large amount of efficient data for identifying an application behavior. Thus, users have to store them in the server.

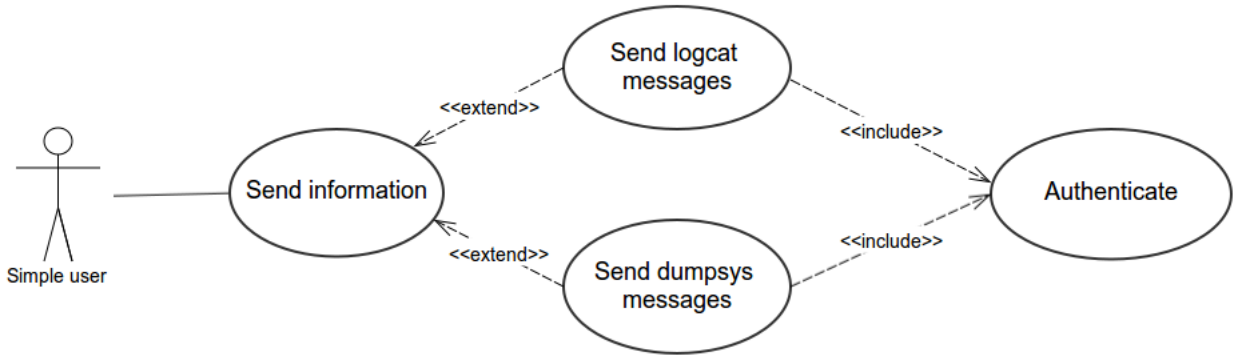


FIGURE 2.2 – Send information use case

Visualize the behavior of a chosen application : detailed use case

After choosing an application to focus on, the user can visualize its behavior. The system allows him to display every single event related to its components. It offers him the possibility to gaze at a single application component and display the type of information he wants to know.

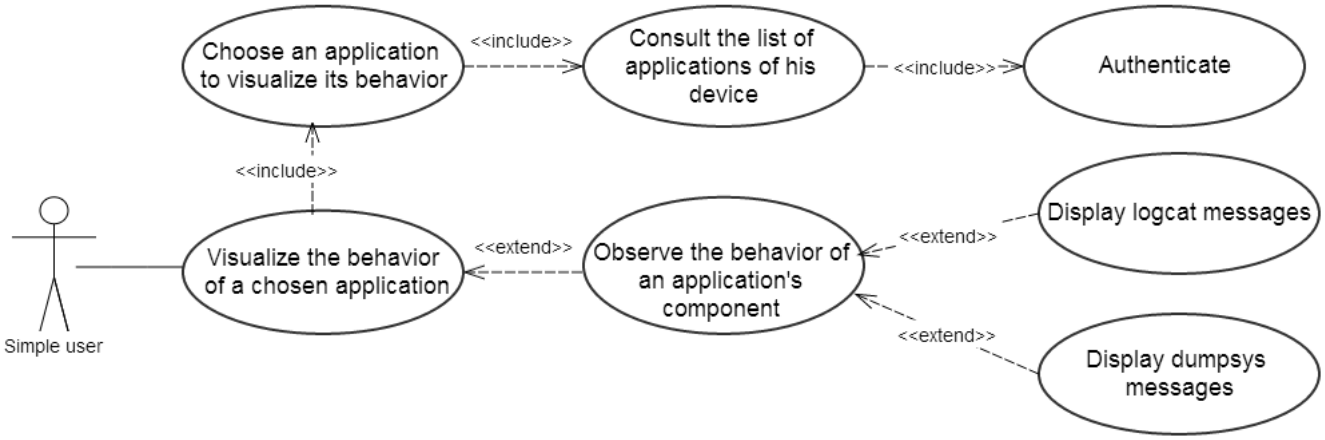


FIGURE 2.3 – Visualize the behavior of a chosen application use case

- **Analyst :**

  Consult the list of applications of a chosen device : detailed use case

  After authentication, the analyst is able to know the list of known users involved in the analysis process through their devices identifiers. By consulting this list, the analyst has the possibility to choose between available devices. Every Android device has a certain number of installed third party applications. Once the device is chosen, the analyst can consult the list of applications available on it.

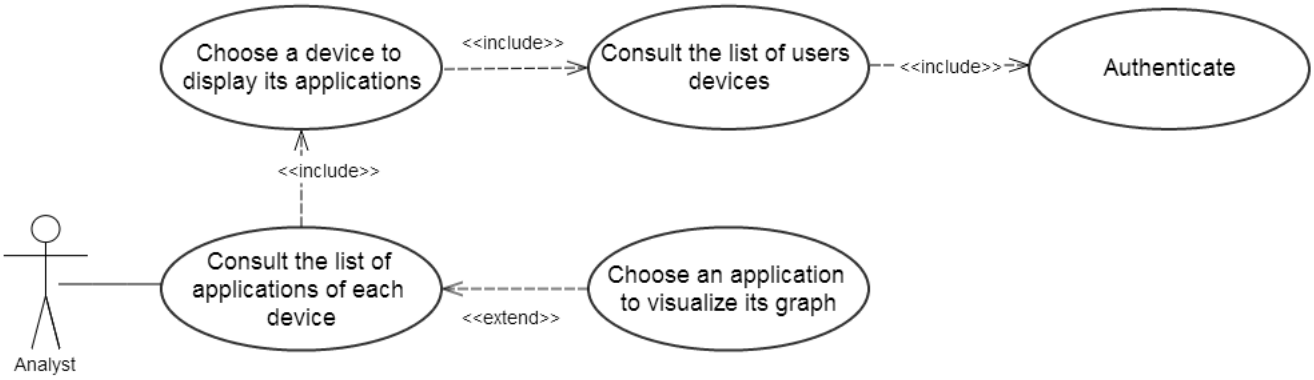


FIGURE 2.4 – Consult the list of applications of a chosen device use case

- **Administrator :**

  Manage analysts : detailed use case

  An administrator is an analyst who manages other analysts. Thus, he should authenticate first in order to do administration tasks. He is responsible for adding analysts by confirming their registrations submissions. He can also delete an analyst to deprive him from getting access to the tool. Moreover, the administrator is able of giving an analyst the administration right.

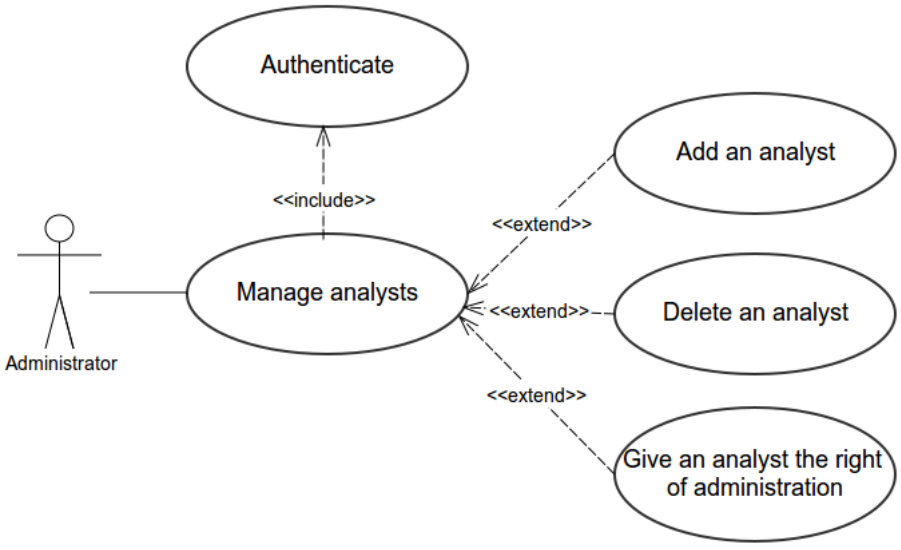


FIGURE 2.5 – Manage analysts use case

2.3 Sequence diagrams :

In this section, we choose some scenarios to present through sequence diagrams in order to bring out the dynamic aspect of the application.

2.3.1 Scenario of application visualization by a simple user :

Figure 2.6 describes a normal scenario of application visualization by a simple user :

Scenario :

1. The scenario begins when the simple user gets access to the tool.
2. The tool displays the homepage.
3. The simple user choose the simple users interface.
4. The tool displays the authentication page of simple users.
5. The simple user authenticates using his device identifier.
6. The system verifies information introduced by the simple user.
7. The system gives the simple user access and displays the list of applications of his device
8. The simple user choose an application to visualize its behavior.
9. The system displays the behavioral graph of the chosen application.

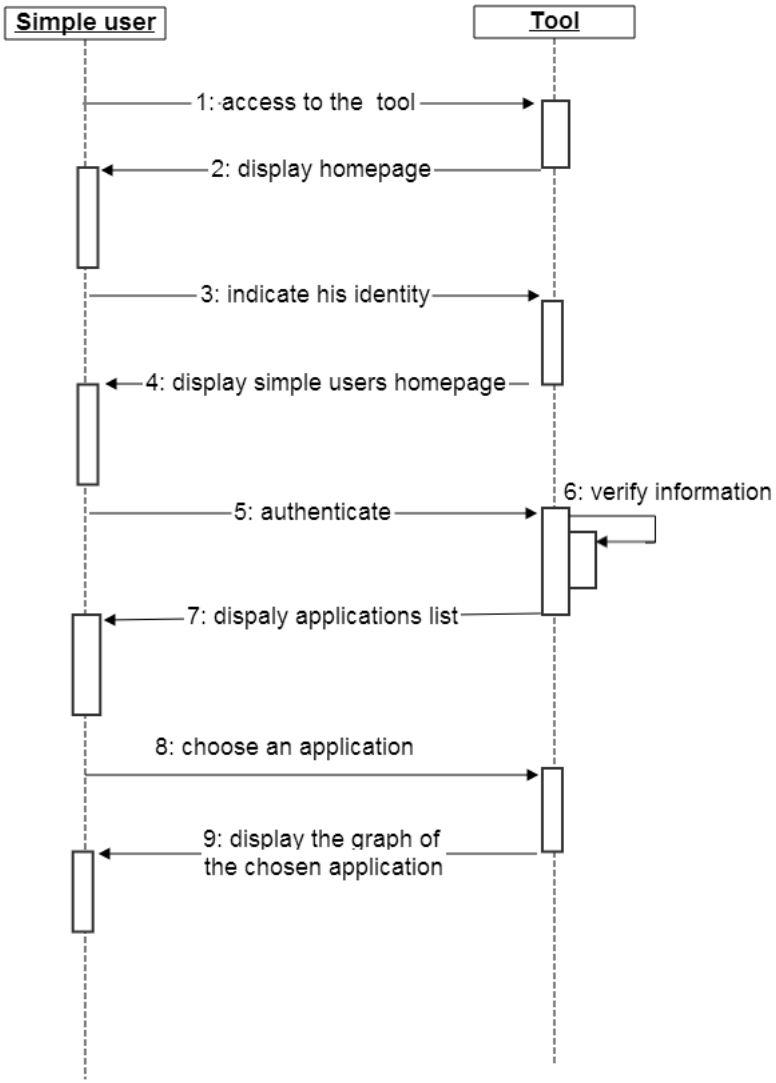


FIGURE 2.6 – Application visualization by a simple user

2.3.2 Scenario of application visualization by an analyst

Figure 2.7 describes a normal scenario of application visualization by an analyst :

Scenario :

1. The scenario begins when the analyst gets access to the tool.
2. The tool displays the homepage.
3. The analyst choose analysts interface.
4. The tool displays the authentication/Registration page of analysts.

5. The user authenticates using his login and his password.
6. The system verifies information introduced by the android user.
7. The system gives the Android user access and displays the list of users involved in the analysis process.
8. The analyst chooses a user's device from the list and asks to consult its applications
9. The system displays the list of application of the chosen device.
10. The analyst chooses an application to visualize its behavior.
11. The system displays the behavioral graph of the chosen application.

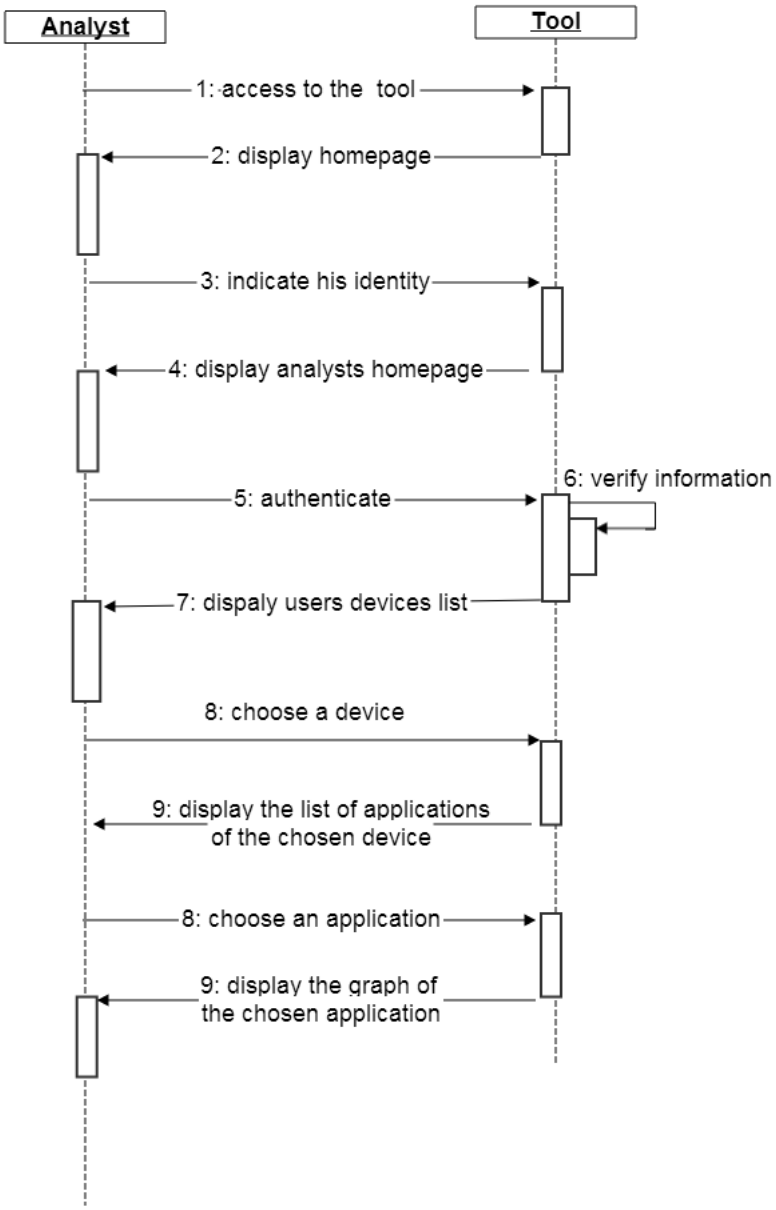


FIGURE 2.7 – Application visualization by an analyst

2.4 Conclusion

Throughout the requirements analysis and the requirements specification presented in this chapter, we have clearly identified the system actors and the functionalities that the product should offer. Indeed, the next chapter will display the global and detailed design of the solution permitting to perform these requirements.

Chapter 3

Design and structure

Having finished and analyzed the specification of our project in the previous chapter, we focus in this chapter on designing a suitable structure for the application. This step is crucial in the course of the project and aims to undertake and prepare the ground for the implementation phase. We begin by exposing the global architecture of our solution. Then, a detailed description of the architecture components is exposed.

3.1 Global design

Since we are working to develop a tool offering users services through visual views, we decided to follow Model-View-Controller(MVC) architecture. This pattern defines a useful separation of concerns which helps developing or changing parts independently.

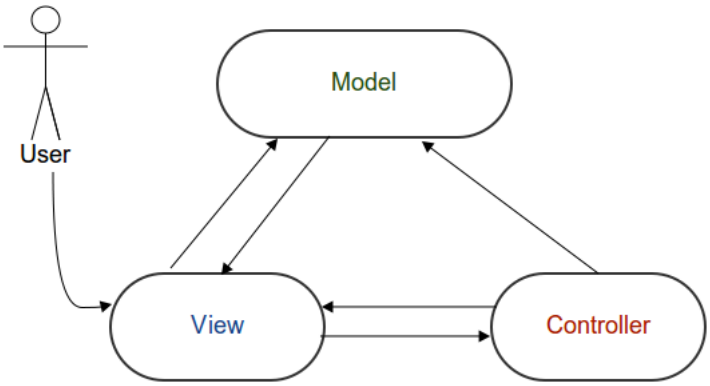


FIGURE 3.1 – MVC pattern

According to the MVC paradigm as shown in the figure 3.1, the View is responsible for defining and rendering the user interface, it is what's presented to the users and how users interact with the application. The Model is where the applications data objects are stored. It represents the underlying data structure of the application. The controller is the decision maker and the glue between the model and view. The controller updates the view when the model changes. It also adds event listeners to the view and updates the model when the user manipulates the view.

The view is responsible for establishing the different interfaces of our tool(registration, authentication, applications, graphs..). The model deals with the storage of necassary data, mainly analysts information and logs messages. The controller controls, manages and ensures interfacing between users, applications and their graphs.

The MVC architecture ensures to our application the following properties :

Maintainability : MVC is designed to be stretchable and evolutionary, and foster the segmentation of responsibilities. It tend to be easier to maintain and thus easier to develop.

Testability : The various logical layers being separated, it is much easier to test them remotely.

3.2 Detailed design

In this section, we start by presenting the architecture of our system. Then we move to detail each layer mentioned above in the global design.

3.2.1 System architecture

Figure 3.2 presents the architecture of our system.

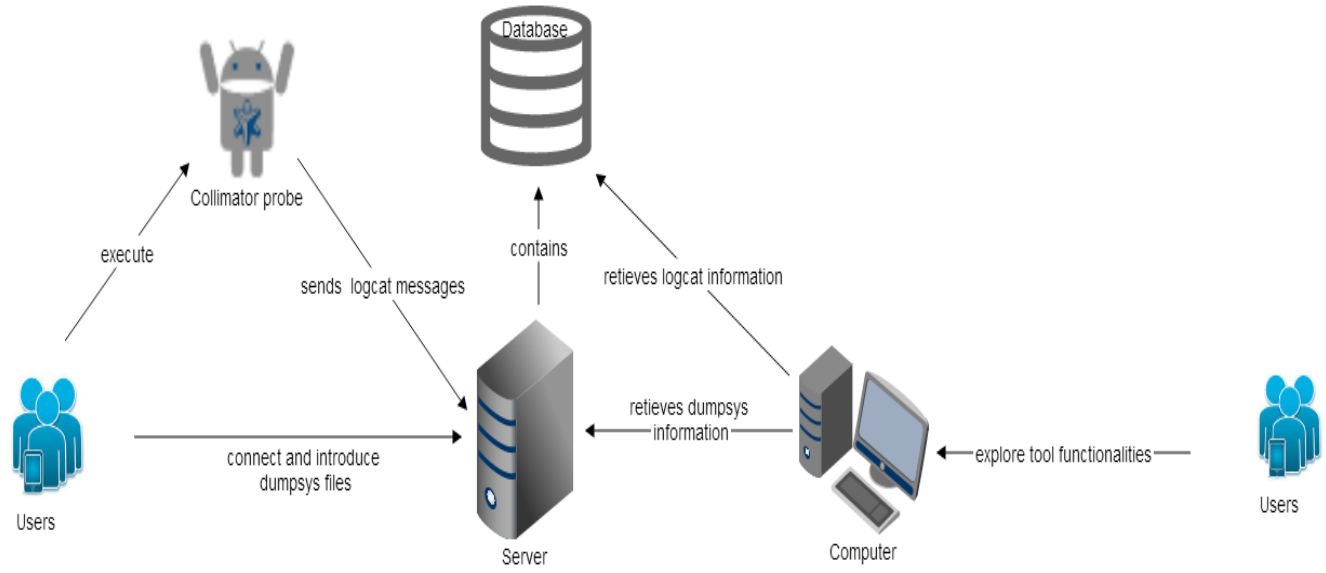


FIGURE 3.2 – System architecture

We explain the schema mentioned above by detailing the different system layers designs.

3.2.2 MVC layers designs

In this section, we start by presenting the data structure that we used in the application. After that, we expose the design of controller functionalities. Finally, we focus on the application's view.

3.2.2.1 Data layer design

In our application, all needed data is centralized in a specific server. Android users have to send logcat and dumpsys messages to this server in order to make them accessible.

As for logcat messages sending, we have taken advantage from « Collimator probe », an android application which was designed and developed by Madynes members. Its main functionalities are collecting these logs and sending them to a server for storage.

The Collimator probe is organized under one basic user Interface, the activity CollimatorActivity, which controls 2 background services :

- LocationService : retrieves geolocation information which is added as an extra string to logcat messages.
- CollimatorService : collects and sends the logcat messages.

Collecting and sending logs were separated. The collected logs are now stored in a buffer before sending them. The reception of each log by the server is done before sending the next log. This has been possible by the use of TCP(Transmission Control Protocol). If an error occurs when when sending, sent data is removed from the buffer, and sending process is restarted a minute later. The following figure presents the collecting and sending processes performed by the Collimator probe :

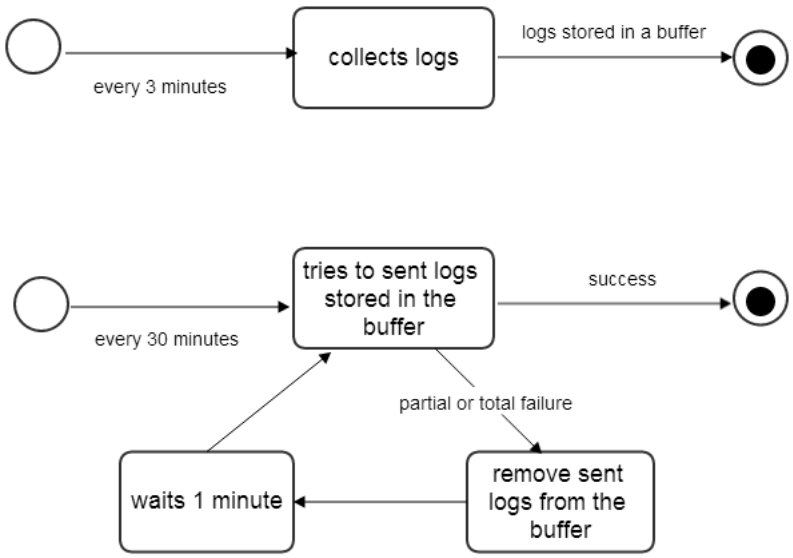


FIGURE 3.3 – Collimator probe functionalities

To store logs sent by users, a dedicated program called Syslog-ng using the data logging standard Syslog has been used for the purpose. It is simply an enhanced syslog daemon which

collects logs from the devices and store them in a MySQL database without any treatment.

The structure of the table containing logs is described by the following table :

Field	Type	Description
id	bigint(20)	Primary key identifying each log
host	var(128)	A unique identifier for android devices
timestamp	var(24)	Invocation date and time of the log
resource	varchar(100)	The application issuing the message
component	varchar(100)	The application's component issuing the log
params	text	The message itself
longitude	varchar(32)	Longitude registered the time of log invocation
latitude	varchar(32)	Latitude registered the time of log invocation
connection	varchar(30)	The connectivity type, namely wifi or mobile

TABLE 3.1 – Logs table

In order to manage analysts, we have dealt with Analyst table which contains specific information about analysts.

Regarding dumpsys information, users have the possibility to access directly to the server via a secure shell (ssh) connection. Then, they have to deposit the contents of dumpsys files in a specific location. After a connection to the server, the tool can access to this location and explore the data stored there.

3.2.2.2 Controller layer design

The controller layer encapsulates the core controller functionalities of the application. There, we explain the controller logic by exposing packages and classes diagrams.

Package diagram :

To move to the design, we rely on the principles of object oriented approach. To this end, we

move from a functional structure through the use cases, to an object structure based on classes and packages.

Given the number of candidate classes defined in our application, it is important to classify them into packages to better understand the overall role of each party and facilitate code maintenance. To identify packages, we relied on two criteria : consistency and independence. Regarding the first criterion, we tried to group classes that have a strong correlation between them and belong to the same functional area. As for the second criterion, we tried to minimize the dependencies between packages by minimizing the number of connections between them.

The figure 3.4 presents the package diagram of our controller :

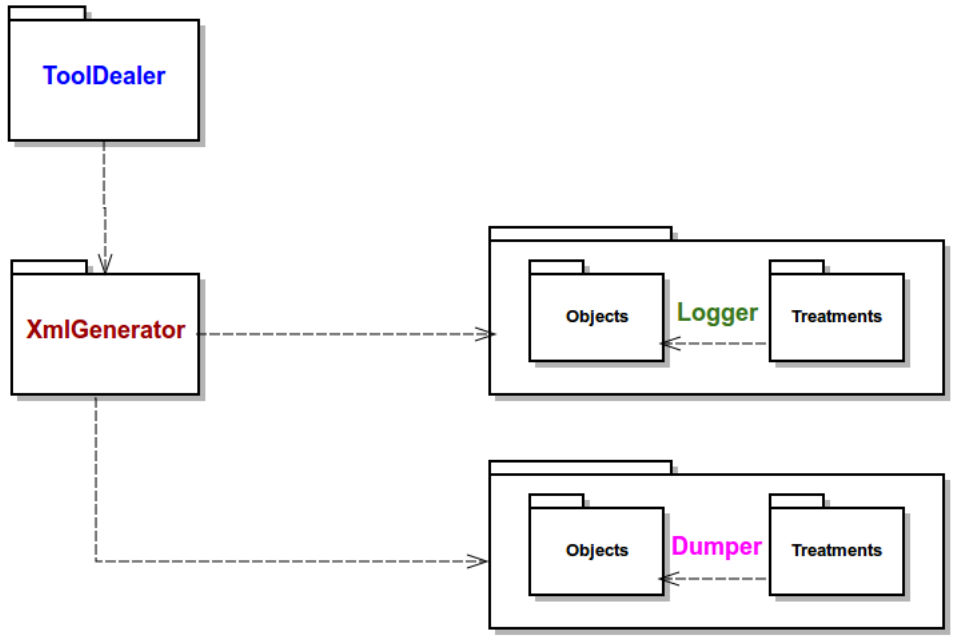


FIGURE 3.4 – Package diagram of the controller

The diagram package for our application shows the decomposition of our system in four packages :

- ToolDealer : it contains classes dealing with providing the tool functionalities.
- XmlGenarator : contains classes dealing wih XML files generation.
- Dumper : contains classes dealing with dumpsys files treatments. It contains 2 packages : Objects which handles the definition of all objects classes and treatments which performs dumping actions.
- Logger : contains classes dealing with logcat messages. It contains also 2 packages : objects for objects classes and treatments for logs manipulation.

Classes diagrams :

The class diagram is a static representation of the structure of a model, namely the classes, the internal structure of classes and their relationships. In what follows, we present a general classes diagram of our controller. Then, we present detailed classes diagrams of packages that we have defined above.

General classes diagram :

The figure 3.6 presents the global classes diagram of our controller :

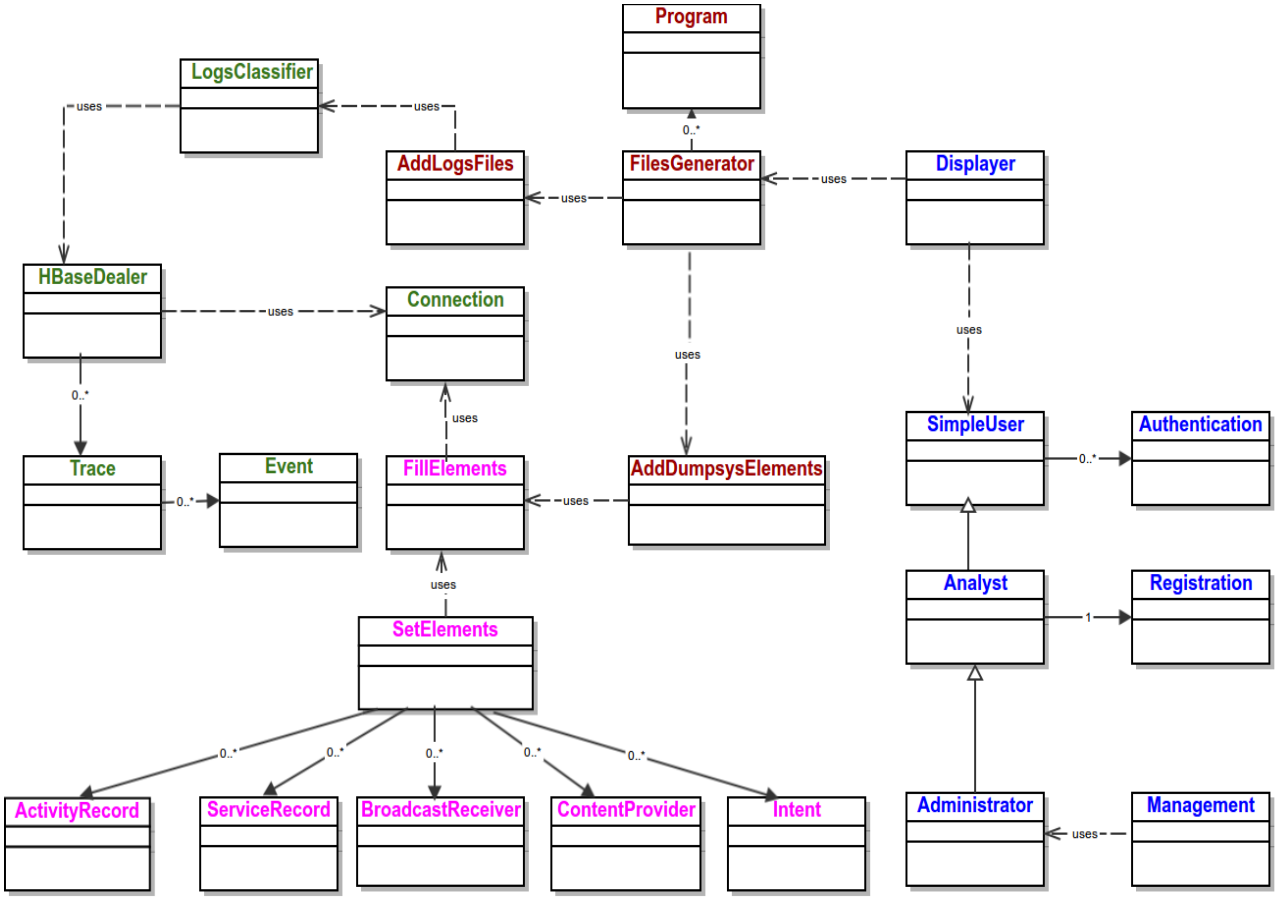


FIGURE 3.5 – Classes diagram of the controller

Logger class diagram :

The package logger is composed of 2 subpackages : Objects and Treatments. In order to deal with logcat data, we have defined the following classes which constitute the Objects subpackage :

- Event : the class presenting the logcat message. It contains a description of different fields that constitute a logcat message.
- Trace : it encapsulates the different events associated to a single user.

As for the subpackage Treatments, it contains 3 basic classes :

- Connection : it ensures the connection to the database where the logs are stored.
- HbaseDealer : it contains basic functions dealing with logcat messages.
- LogsClassifier : it classifies logcat messages into categories. The categories are the basic components of an android application : activities, services, broadcast receivers, content providers and intents.

The class diagram of figure 3.6 describes relationships between these classes :

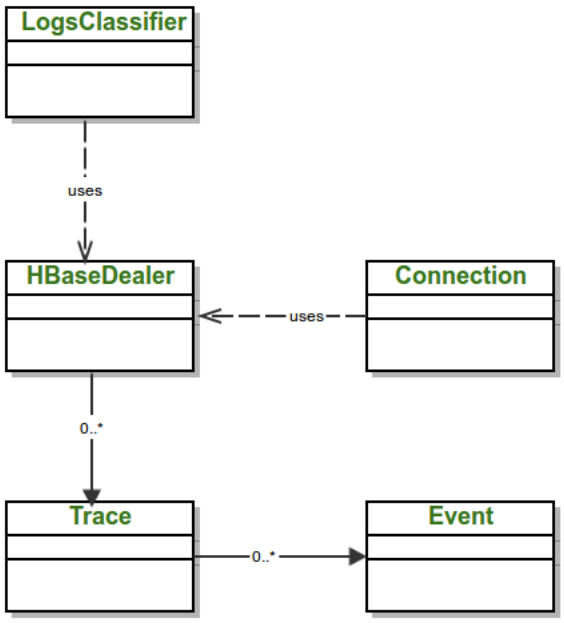


FIGURE 3.6 – Logger class diagram

Dumper class diagram :

Like the logger package, the dumper package is composed of Objects and treatments subpackages. Figure 3.7 illustrates the dumper package class diagram :

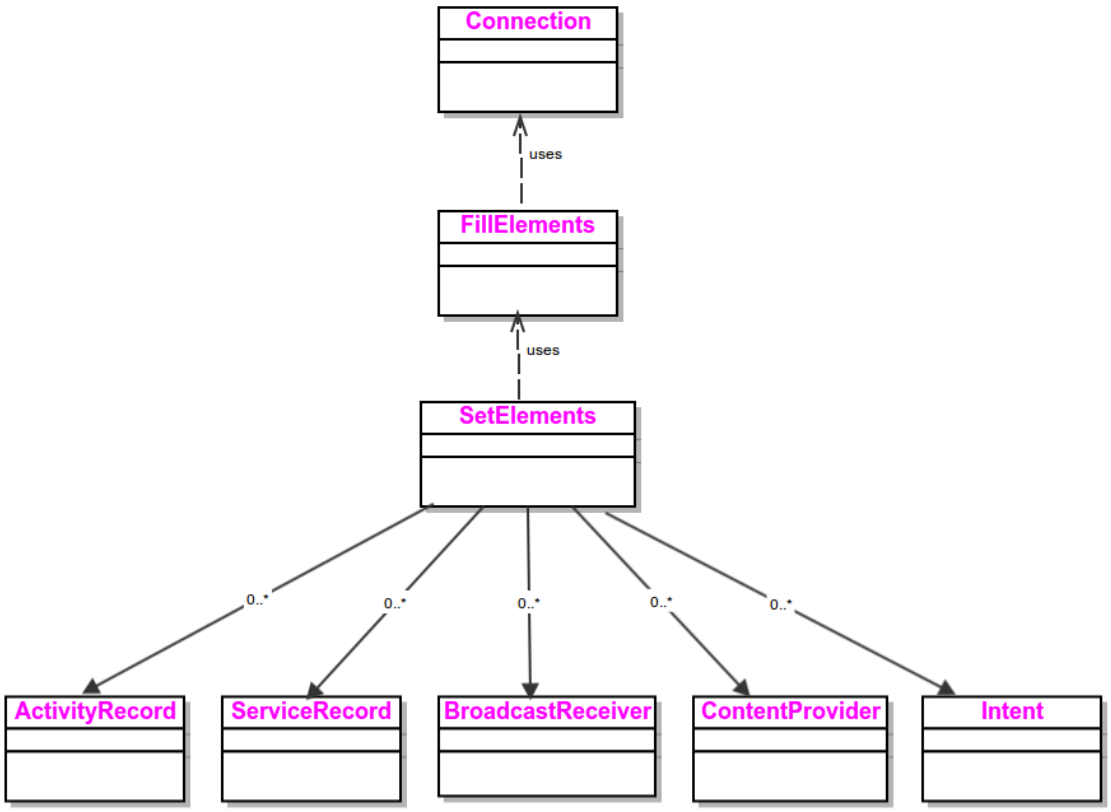


FIGURE 3.7 – Dumper class diagram

The subpackage Objects contains all objects classes needed to classify dumpsys information into android components categories. We have defined the following classes :

- ActivityRecord : it defines the Activity component of an android application.
- Intent : it defines a dumpsys information dealing with intents.
- Service Record : it is a service registered in the dumpsys file.
- ContentProvider : it defines the Content provider component of an android application.
- BroadcastReceiver : it's the class which deals with broadcast receivers.

The subpackage Treatments contains the following basic classes :

- Connection : it ensures the connection to the dumpsys files location in the server
- SetElements : it sets dumpsys information for each android application component
- FillElements : it fills dumpsys classes with their associated attributes.

XmlGenerator class diagram :

The package XmlGenerator uses Dumper and Logger packages in order to structure data into Xml files having a specific format. Each application of each user is presented by an xml files containing logcat and dumpsys information.

Figure3.8 illustrates the XmlGenerator class diagram :

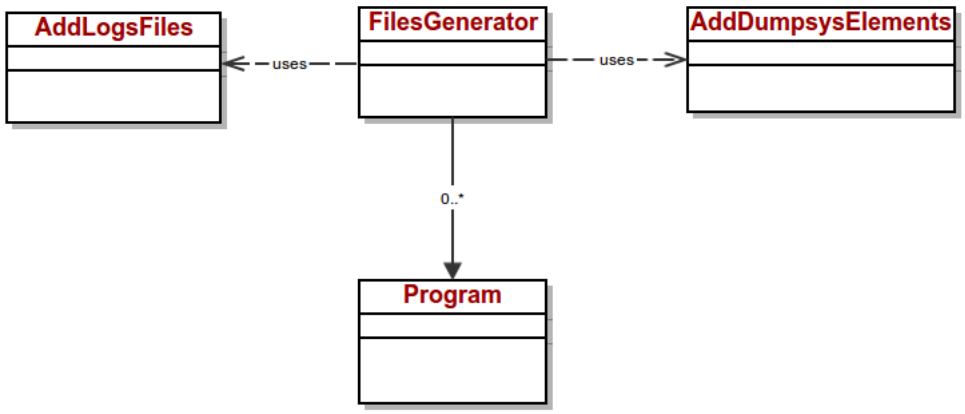


FIGURE 3.8 – XmlGenerator class diagram

As shown in the figure above, XmlGenerator contains 4 classes :

- Program : it refers to an xml file of a given application of a given user
- AddLogsElements : it deals with the structuring of logs information in xml files
- AddDumpsysElements : it deals with the structuring of dumpsys information in xml files
- FilesGenerator : it focuses on xml files generation.

ToolDealer class diagram :

The ToolDealer contains all classes providing the different services offered to the differents types of users. It contains 7 classes :

- SimpleUser : the class presenting the simple user actor.
- Analyst : the class presenting the analyst actor.
- Administrator : the class presenting the administrator actor.
- Authentication : it deals with the authentication of different types of users.
- Registration : it focuses on the registration process for analysts.
- Displayer : it contains functions dealing with the extraction of the convenient devices and applications.
- Management : it refers to the management tasks performed by the administrator

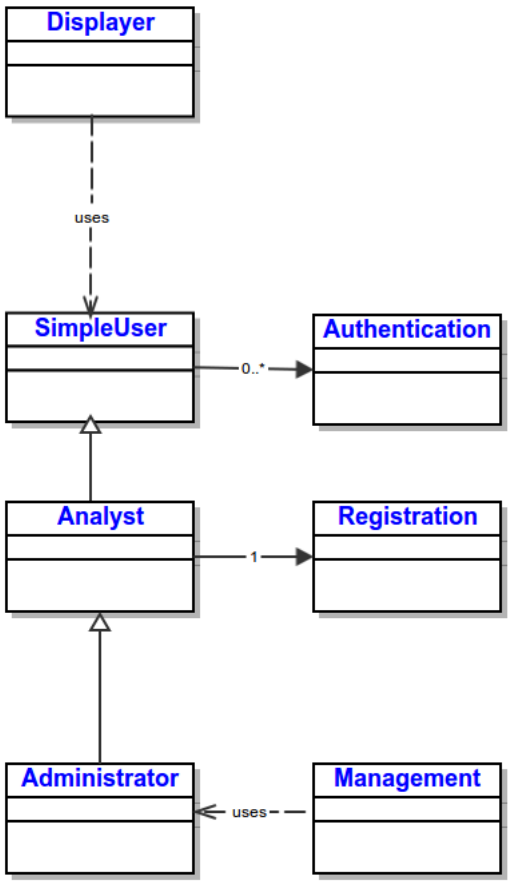


FIGURE 3.9 – ToolDealer class diagram

3.2.2.3 View layer design

The view is assimilated to the tool presentation. The tool's view is characterized by the class diagram in the figure 3.10

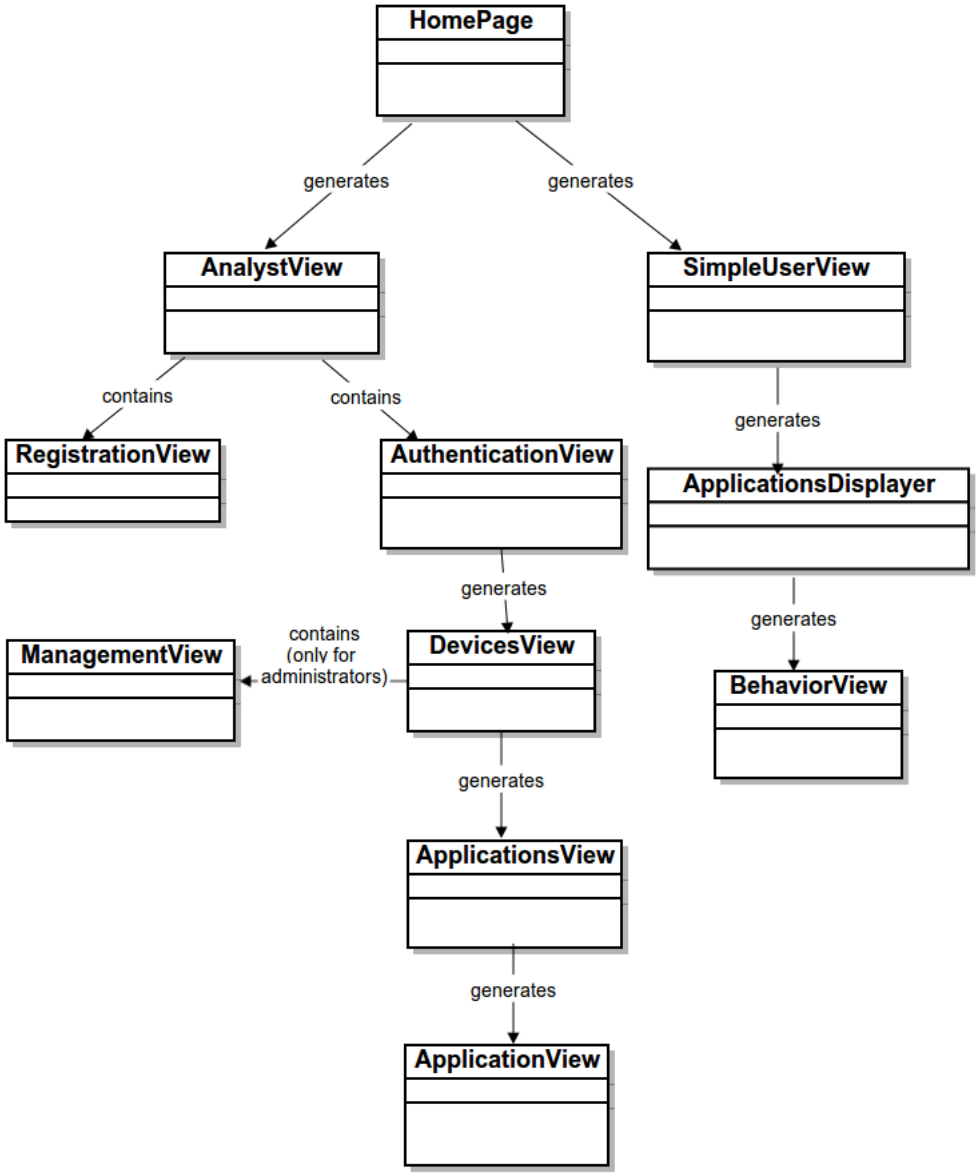


FIGURE 3.10 – View class diagram

The first class of this diagram is the HomePage class. Depending on the type of user, it generates the suitable view. TheAnalystView represents the home page for analysts. It contains the registration and authentication views for analysts .The DevicesView class

represents the view giving the analyst the possibility to consult users list and choose between them. If the analyst is an administrator, this view contains a ManagementView permitting him to perform administration tasks. The ApplicationsView displays the list of applications executed in the device of the user chosen in the previous view. The ApplicationView is responsible for representing the graph of a single application chosen by the analyst. For simple users, they can get access to the tool via an authentication view described by the SimpleUserView. The ApplicationsDisplays deals with displaying the list of applications of the authenticated user. The BehaviorView figures out the graph of one application of simple user's applications.

3.3 Conclusion

In this chapter, we presented the general and detailed design of our solution to facilitate the transition to the implementation phase. In the next chapter, we will expose the work realized throughout this project as well as the environment which we will use to develop it.

Chapter 4

Implementation details

This chapter discusses the implementation of the applications components. We begin by presenting the environment on which we worked. Then, we present the development process. Finally, we depict some interesting interfaces of our application.

4.1 Work environment

In this section, we will present the hardware and software environment on which we have done our project.

4.1.1 Hardware environment

Our tool was developed on a PC having the following characteristics :

- Operating system : Linux (distribution Ubuntu).
- Processor : Intel Core 2 Duo.
- RAM : 3 Go.

In order to perform its basic tasks, the tool must have access to a distant server where different information are stored.

We have also used Android-based devices, mainly smartphones and tablets in order to depict Android applications behaviors.

4.1.2 Software environment

JAVA :

Java is an object-oriented language. It is centered on creating objects, manipulating them, and making them work together. This allows to create modular programs and reusable code. Moreover, Java puts a lot of emphasis on early checking for possible errors, since Java compilers are able to detect many problems that would first show up during execution time in other languages.One of the most significant advantages of Java is its ability to move easily from one computer system to another. In fact, Java succeeds at this by being platform-independent at both the source and binary levels and thus allowing to run the same program on many different systems. [N12]
Because of Javas modularity, robustness and cross-platform capabilities, it has become a language of choice for providing our solution.

Eclipse Indigo :

Eclipse IDE is a free development environment with potential to create development projects implementing several languages, mainly Java language. The specifity of Eclipse IDE is that its architecture is fully developed around the concept of plug-in. This means that all of its features are developed as plug-in. In our project it helped in the development task of our tool.[N13]

HBase :

HBase is a database management system. It is modeled after Google BigTable and is part of the world's most popular big platform, Apache Hadoop. As the open source implementation of Google's BigTable architecture, HBase scales to billions of rows and millions of columns [N14]. Thus, it is well suited for big data sets, which make it, in our application, a great solution for real-time ingestion of log data.

Running on top of Hadoop Distributed File System(HDFS), HBase ensures that write and read performance remain constant. In fact, HBase augments HDFS by providing record-based storage that allows users and applications to perform fast, random reads and writes to data. Changes are cataloged in memory and eventually pushed down to HDFS for persistence [N15]. This enables the Hadoop system to serve random reads and writes

to users and applications across big tables in real time. This feature ensures an efficient response time when dealing with HBase tables.

HBase is a column oriented database which stores its contents by column rather than by row. Instead of retrieving a record or row at a time, an entire column is retrieved and thus it becomes very powerful and efficient since data analytics are usually concerned with only one field or column of a record. The access becomes much faster, and much more relevant data can be extracted from the database in a shorter period of time. Moreover, HBase allows for many attributes to be grouped together into what are known as column families.

4.2 Development process

In this section, we detail the different steps of our project's process.

4.2.1 Logs analysis

In order to make logs messages understandable for users, we proceed with an analysis phase. For each log record, we associate a list of attributes with meaningful names. In fact, we figure out the known attributes. Then, we decompose the long ones. Moreover, we add other attributes related to logs senders (devices identifiers, location, type of connection...)

Then, to store this hierarchical data, we have moved from MySQL to HBase. The following pseudo code does this conversion.

Algorithm 1 FromMySQLToHBase

Require: $LogsTable$

Ensure: $ConversiontoHBase$

$connection(LogsTable)$
$applications \leftarrow getApplications()$
for $i = 1$ to $applications.size()$ **do**
  $tuples \leftarrow getTuples(applications.get(i))$
  for $k = 1$ to $tuples.size()$ **do**
    $ListParams \leftarrow getFamilyColumns(tuples.get(k)).params)$
    $insertionHBase(k, ListParams)$
  end for
end for

After connceting to MySQL table, we get the list of applications via getApplications() method. Then, for each application, we get the list of tuples dealing with it. We remplace the attibute params which represents the content of the log record by a family of columns via getFamilyColumns(). Finally, we insert the record in the HBase.

This is an example of a log message given by the logcat tool.

07-02 09 :27 :40.405 1806 2012 I am_destroy_activity :
[111149 ,200, com.PinballGame/. PinballGameActivity]

After analyzing it and identifying its different attributes, this record is stored in the HBase as follows :

com.PinballGame00541 column=component :, timestamp=1372763573135, value=am_destroy_activity
com.PinballGame00541 column=datetime :, timestamp=1372763573130, value=2013-07-02 09 :27 :40.405
com.PinballGame00541 column=host :, timestamp=1372763573129, value=d8249c8749018
com.PinballGame00541 column=latitude :, timestamp=1372763573132, value=48.6688354
com.PinballGame00541 column=longitude :, timestamp=1372763573133, value=6.1621055
com.PinballGame00541 column=params :Component Name, timestamp=1372763573135 ,

value=PinballGameActivity
com.PinballGame00541 column=params :Reason, timestamp=1372763573139, value=finish-imm
com.PinballGame00541 column=params :Task ID, timestamp=1372763573137, value=200
com.PinballGame00541 column=params :Token, timestamp=1372763573136, value=111149

The component column refers to the type of the event. The datetime gives the invocation date and time of the log. Params is a family of columns giving details about the content of the log. We find in this record additional attributes. The attribute host refers to the device identifier of the sender. The attributes latitude and longitude depict the location information of the sender. These information are given by Collimator Probe, the probe used to send logs messages. The timestamp refers to the date of creation of records.

4.2.2 Logs visualization

The visualization was done via the treemapping technique. A Treemap, also known as Heatmap, is an important tool for representing and visualizing a lot of information at a glance. A treemap displays hierarchical relationships through a set of rectangles, sized proportionately to each data point, clustered together into one large rectangle. The rectangular screen space is divided into regions, and then each region is divided again for each level in the hierarchy. Treemaps show part-to-whole relationships with each rectangle in the treemap representing a category from the dataset. The nested regions show hierarchical relationships and allow for quantitative comparisons of attribute values [N16].

In order to integrate the treemap in our application, logging information about each application of each user have been gathered in Xml files having the following format :

< ?xml version='1.0' encoding='ISO-8859-1' ?>
< !ELEMENT root (label,(branch | leaf)*)>
< !ELEMENT branch (label,(branch | leaf)*)>
< !ELEMENT leaf (label,weight,value) >
< !ELEMENT label (PCDATA) >
< !ELEMENT weight (PCDATA) >

< !ELEMENT value (PCDATA) >

Then, we have developed an appropriate algorithm of treemapping. The following pseudo code describes the basic instructions of our algorithm :

Algorithm 2 Treemapping

Require: $fileName$

Ensure: $treemap$

$tree \leftarrow BuildXML(fileName)$

$root \leftarrow tree.getRoot()$

$treemap \leftarrow initialize(root)$

$treemap.setProperties()$

return $treemap$

After parsing the xml file, the buildXml function returns the corresponding tree to the file's hierarchy. Then, we extract the root of the tree via the getRoot() method. The initialize method constructs the treemap. In fact, it's responsible for identifying the differents elements of the hierarchy, calculate their positions and drawing them. Via setProperties() method, we set different properties of the treemap like size, color or zooming feautures.

4.2.3 Graphs interpretation

In this subsection, we will analyze some of the obtained applications graphs.

Paintball (Game) :

In the graph of figure 4.1, we have one block represented. In fact, this game is based only on activities. In fact, like a major part of games applications, this application focuses on the interaction with users by providing interfaces.

FIGURE 4.1 – Paintball graph

Facebook :

Facebook is an application based on a strong system of notifications. Thus, the most important part in the graph of figure 4.2 is the block of services. An example of service component in Facebook application would be the friend request notifications. They would continue to run, even if we switch to another activity or application.

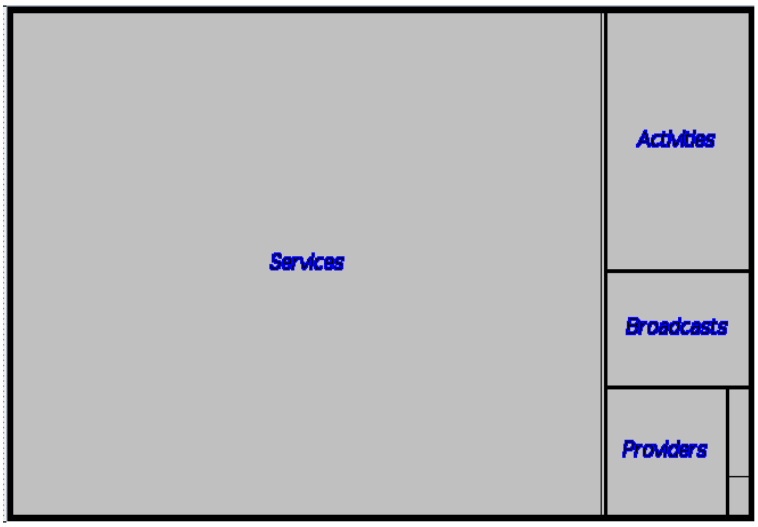


FIGURE 4.2 – Facebook graph

Viber :

Viber is an application for free telephony calls. As shown in the graph of figure 4.3, it uses a considerable number of activities in order to offer its services. We find mainly, but in lower number, some intents, services and broadcasts registered by the logging system.

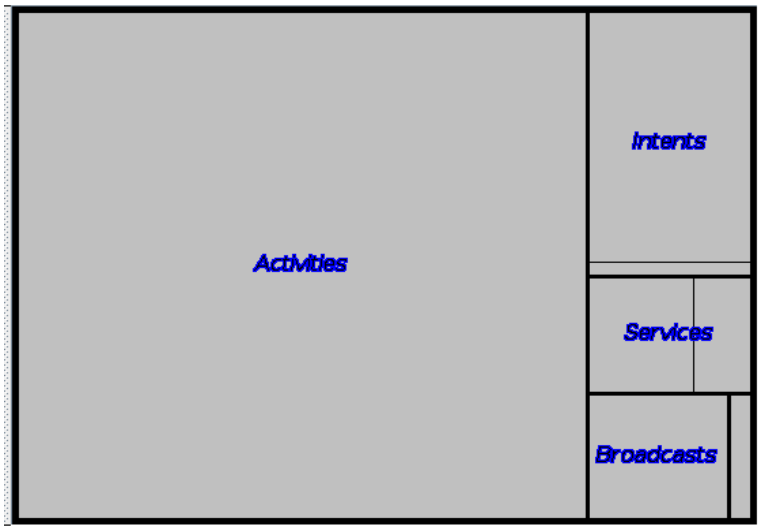


FIGURE 4.3 – Viber graph

Android telephony :

Android telephony is a system application which interacts with a huge number of applications. This is can be concluded by the presence of many messages related to the broadcast receivers. Figure 4.4 depicts the telephony graph :

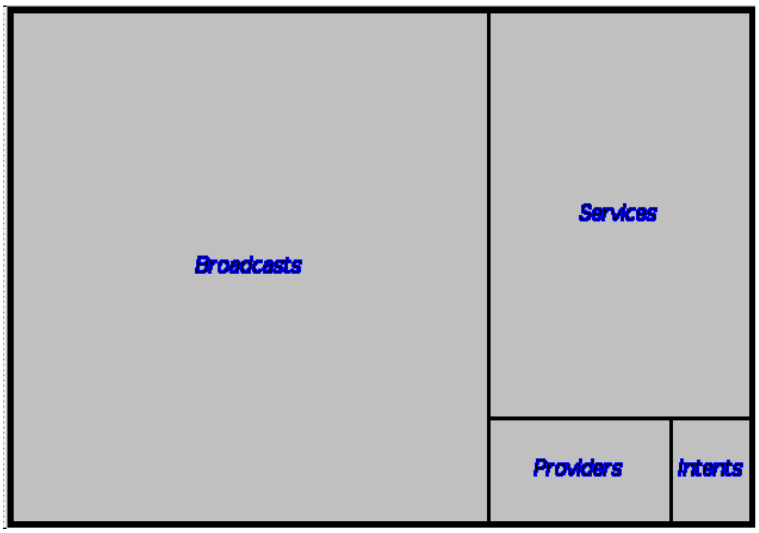


FIGURE 4.4 – Telephony graph

4.3 Tool screenshots

In this section, we depict the achieved work through some screenshots of the application.

Home page :

The user has to mention if he wants to get access to the tool as a simple user or an analyst. Thus, figure 4.5 allows him to do that.

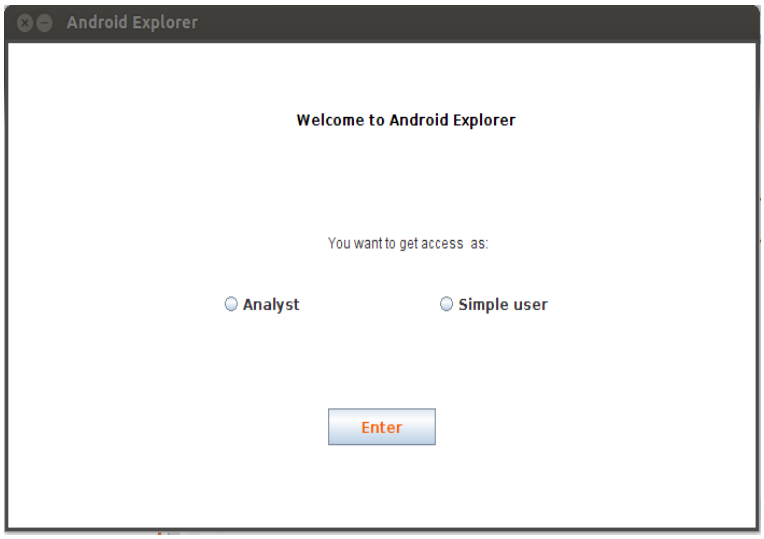


FIGURE 4.5 – Home page

Analysts home page :

If he is an analyst, a Registration/Authentication view, as shown in figure 4.6, will be displayed.

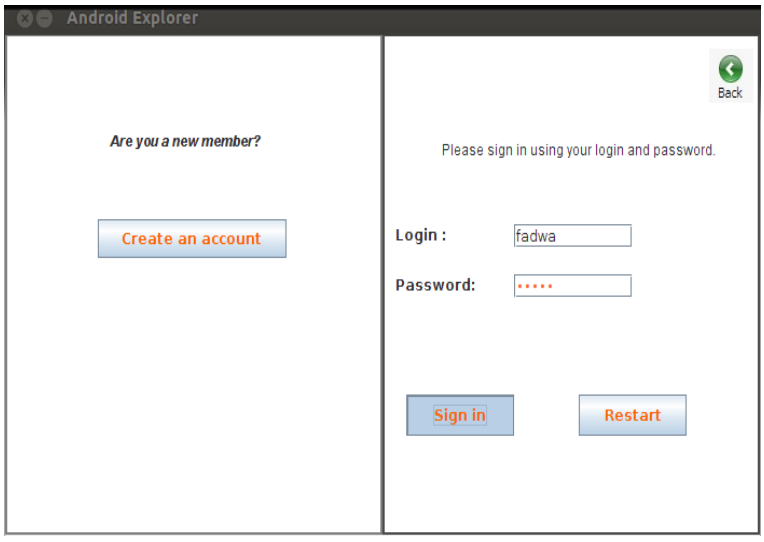


FIGURE 4.6 – Analyst home page

Analysts registration :

If he is a new analyst, he should register before getting access to the tool functionalities. The interface presented by of figure 4.7 permits him to submit his registration :

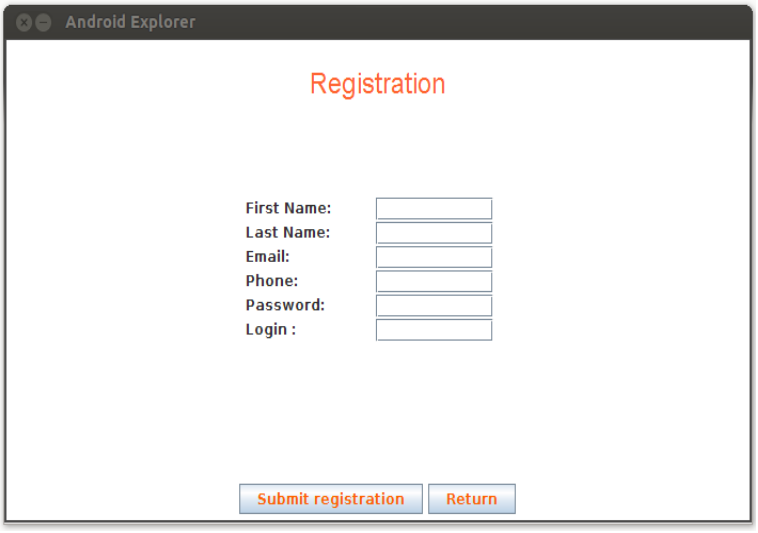


FIGURE 4.7 – Analyst registration interface

Devices view :

As shown in figure 4.8, the analyst, once authenticated, can consult the list of registered devices and choose one of them in order to explore it.

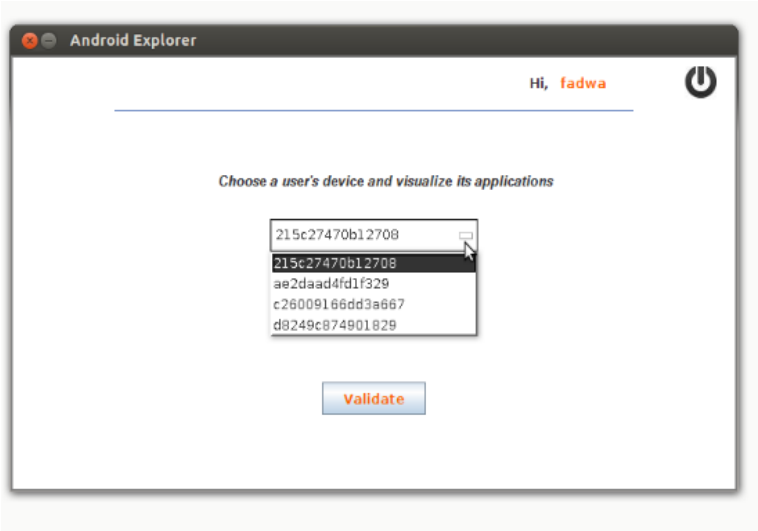


FIGURE 4.8 – Devices view

Applications view :

Once the device is chosen, the interface of figure 4.9 will be displayed to give him the possibility to consult the list of applications of that device and choose one of them in order to visualize its behavior.

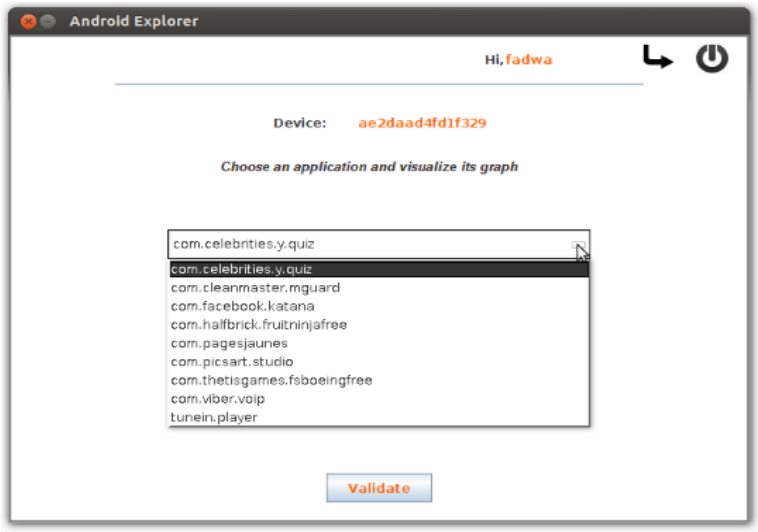


FIGURE 4.9 – Applications view

Application behavior graph :

After choosing the application, the analyst can visualize its behavior. The interface of the figure 4.10 permits him to get different information of different components of the given application.

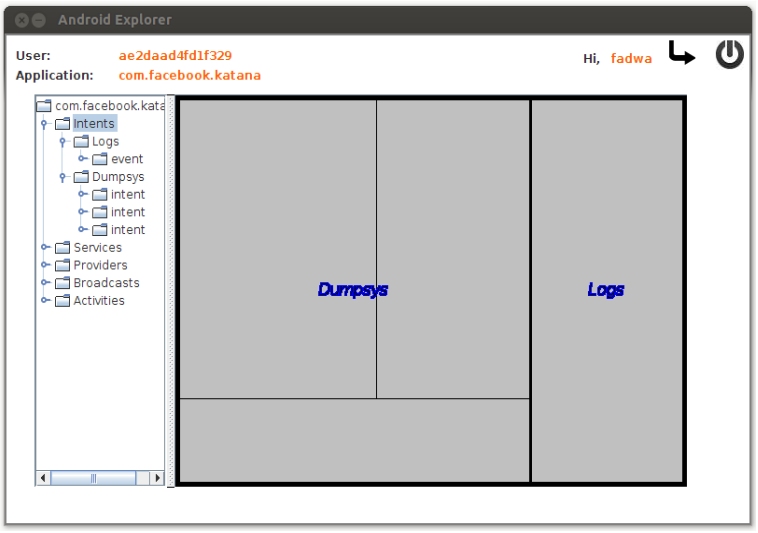


FIGURE 4.10 – Application behavior graph

Logcat information interface :

The analyst can focus on a single component and get its logcat information. By clicking a single rectangle or choosing from the tree hierarchy, the analyst can have details about a logcat event related to the chosen component. (figure 4.11)

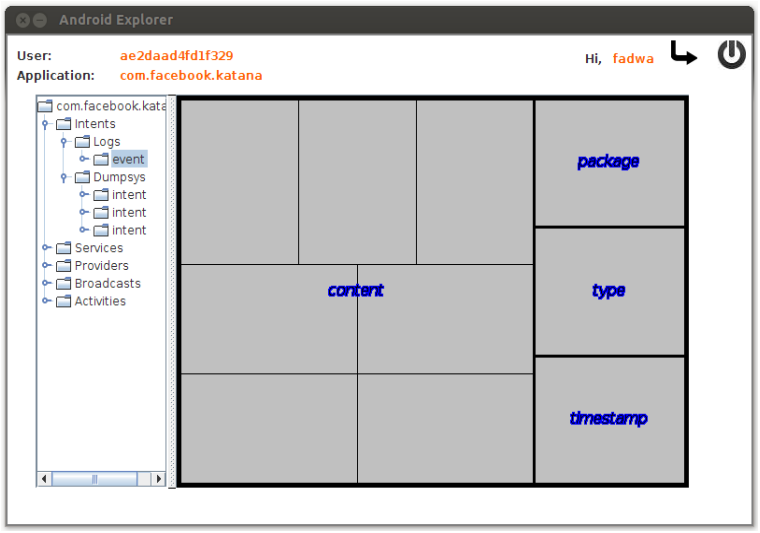


FIGURE 4.11 – Logcat information interface

Logcat attributes view :

To get attributes values of the chosen element, the analyst has to click one of the set of rectangles representing that element. (figure 4.12)

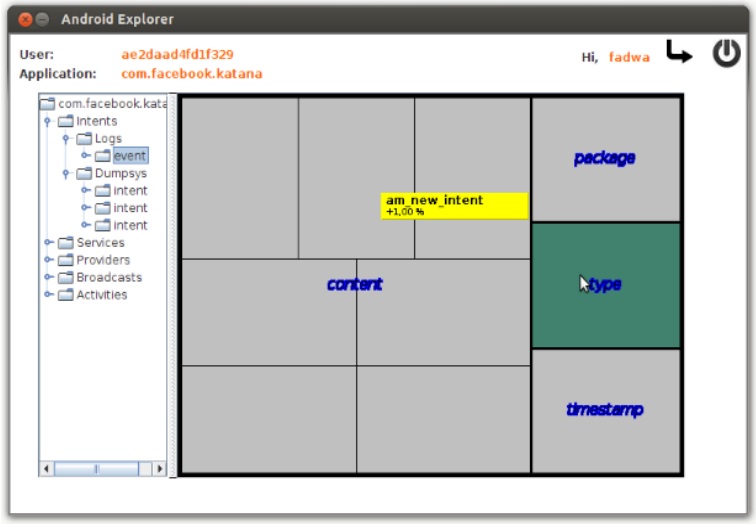


FIGURE 4.12 – Logcat attributes view

Dumpsys information view :

The analyst can also get dumpsys information about a specific component. Figure 4.13 represents the structure of dumpsys information related to the intents of Facebook application :

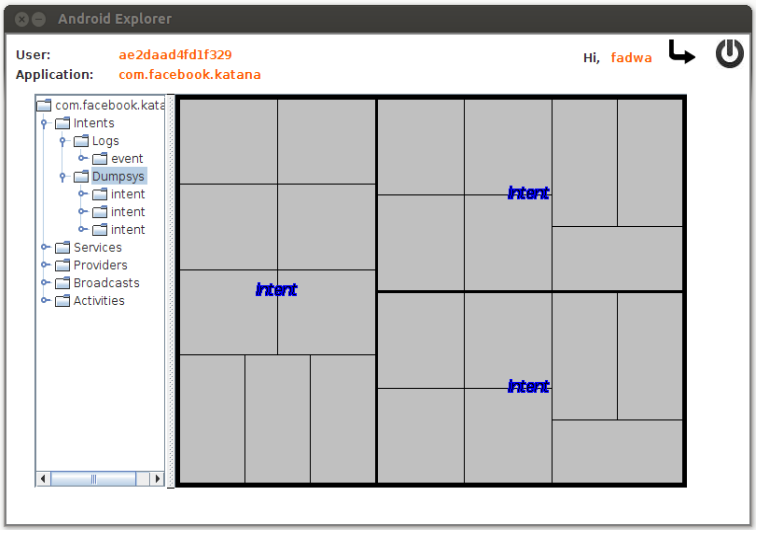


FIGURE 4.13 – Dumpsys information view

Dumpsys attributes view :

The analyst can get different attributes associated to a specific element of the hierarchy. As shown in figure 4.14, he can choose the attribute and click the rectangle associated to it.

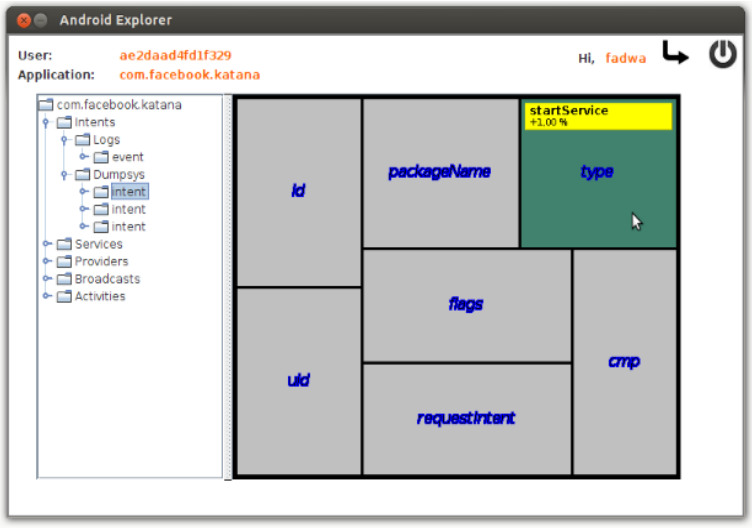


FIGURE 4.14 – Dumpsys attributes view

Zooming view :

The visualization can be interactive allowing drilling down into some section of the hierarchy and visualizing it. As shown in figure 4.15, the system offers the possibility of zooming in and out.

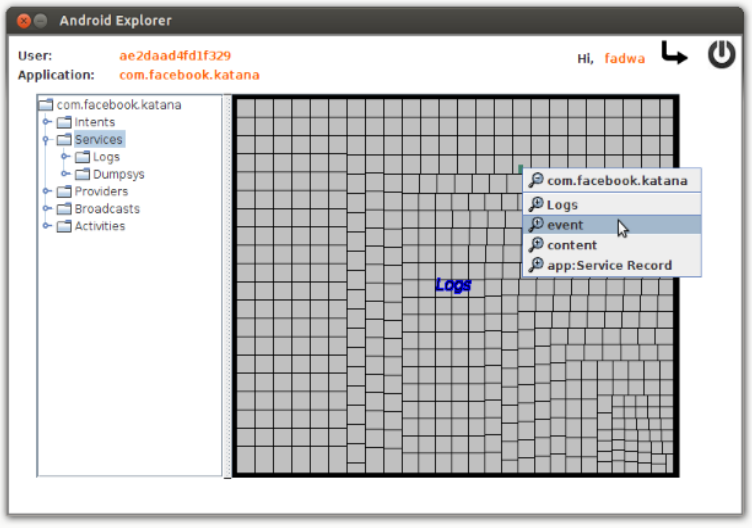


FIGURE 4.15 – Zooming view

Administrator interface :

Administrators are analysts who have the additional task of managing analysts. They can get access to the administration view through the interface of figure 4.16 :

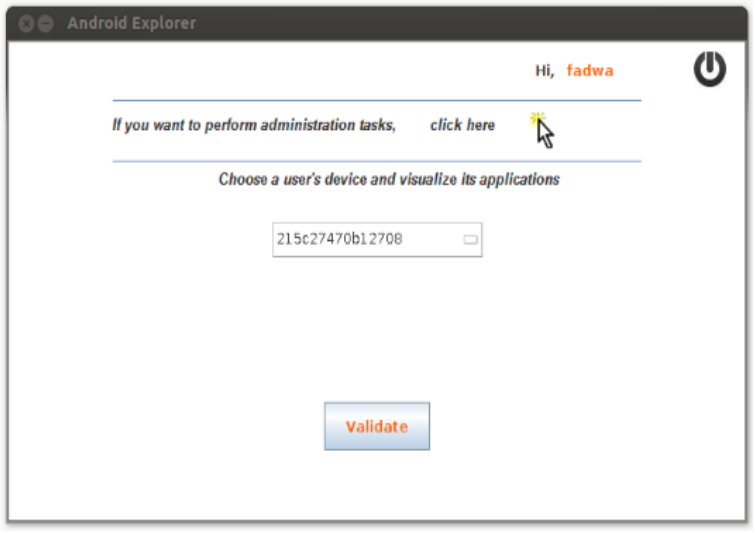


FIGURE 4.16 – Administrator interface

Management view :

Administrators can confirm the registrations submissions of analysts. They have also the possibility to give them the administration right or deleting them completely. Figure 4.17 represents the interface permitting them to perform these tasks :

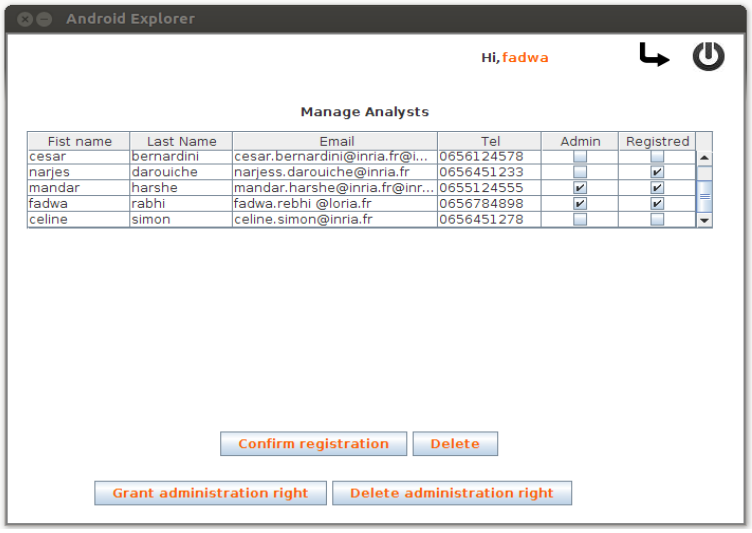


FIGURE 4.17 – Management view

Simple user homepage :

For simple users, the authentication is done via their device identifiers. So, to log in, the simple user has to enter his device identifier through the interface shown in figure 4.18 :

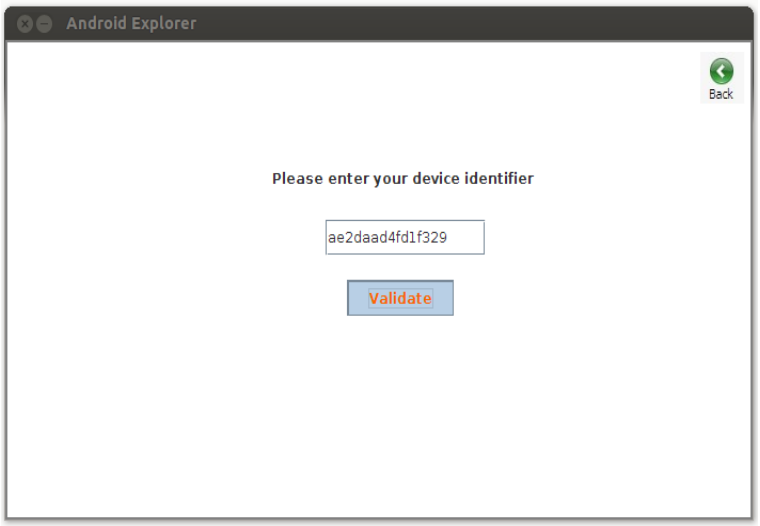


FIGURE 4.18 – Simple user homepage

Simple user applications view :

Once authenticated, the simple user gets access the list of the applications executed in his device. Figure 4.19 depicts the interface giving him the possibility to choose which application to visualize.

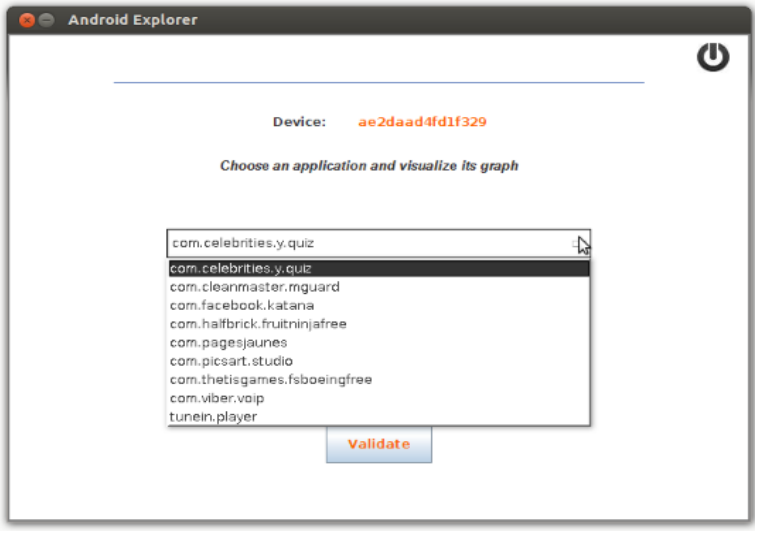


FIGURE 4.19 – Simple user applications view

Simple user application graph :

The interface 4.20 gives the simple user the possibility to visualize the graph of the chosen application :

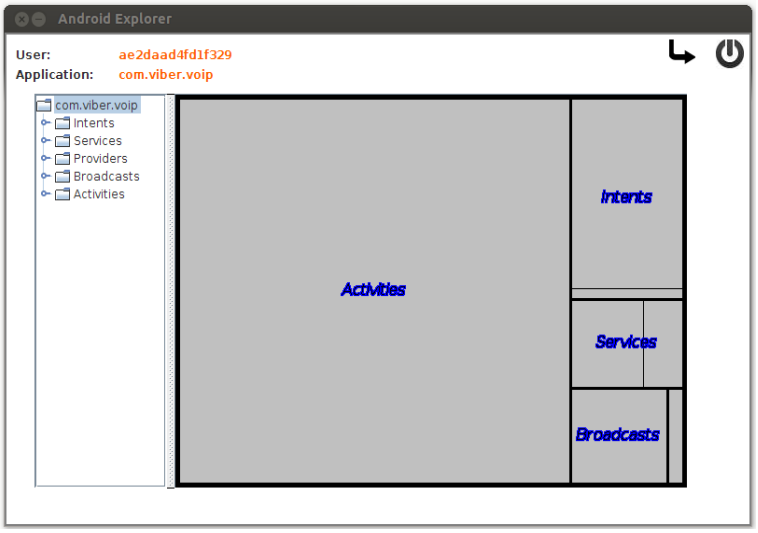


FIGURE 4.20 – Simple user application view

4.4 Conclusion

In this chapter, we presented the hardware and development environment in which we created our application. We subsequently presented the most significant steps of our development process. Finally, we depicted some interfaces of our application.

Bibliography

[B1] Android application development, Reto Meir, WILEY, 2009.

Netography

[N1] http ://www.inria.fr/en/institute/inria-in-brief/inria-in-a-few-words <29/03/2013>

[N2] http ://www.loria.fr/la-recherche-en <29/03/2013>

[N3] http ://resources.infosecinstitute.com/android-architecture-and-pen-testing-of-android-applications <29/03/2013>

[N4] http ://www.infoq.com/articles/detection-of-mobile-malware <02/04/2013>

[N5] http ://www.edureka.in/blog/android-interview-questions-answers-for-beginners <20/04/2013>

[N6] http ://msdn.microsoft.com/en-us/library/ff512738(v=office.14).aspx <20/04/2013>

[N7] http ://elinux.org/Android_Logging_System<20/04/2013>

[N8] http ://android-test-tw.blogspot.fr/2012/10/dumpsys-information-android-open-source.html <25/04/2013>

[N9] http ://h30097.www3.hp.com/docs/base_doc/DOCUMENTATION/V51B_HTML/MAN/MAN8/0135___.HTM <25/04/2013>

[N10] http ://code.google.com/p/androguard <01/06/2013>

[N11] http ://www.honeynet.org/taxonomy/term/188 <01/06/2013>

[N12] http ://www.webdotdev.com/nvd/articles-reviews/java/java-advantages-and-disadvantages-1042.html <29/06/2013>

[N13] http ://blog.zenika.com/index.php ?post/2011/06/26/Eclipse-Indigo-et-EGit-1.0.0 <15/07/2013>

[N14] http ://blog.safaribooksonline.com/2013/04/26/hbase-an-opensource-bigtable-database <15/07/2013>

[N15] http ://www.cloudera.com/content/cloudera/en/products/cdh/hbase.html <15/07/2013>

[N16] http ://betterevaluation.org/evaluation-options/treemap <15/07/2013>

Appendix

HBase guide

Hbase installation

- To setup a standalone HBase instance, hadoop and hbase installation are surely needed.
- Just like Hadoop, HBase requires at least Java 6 from Oracle.
- To have a running standalone HBase instance, we have to start it via the shell.

```
$./bin/start-hbase.sh
starting Master, logging to logs/hbase-user-master-example.org.out
```

Hbase shell

- After starting HBase, we can connect to your running HBase via the shell.

```
$ ./bin/hbase shell
HBase Shell ; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version : 0.90.0, r1001068, Fri Sep 24 13 :55 :42 PDT 2010
hbase(main) :001 :0>
```

- Create permits to create tables. Put is used to insert rows. In the following example, we create a table named test with a single column family named cf and we insert some values.

```
hbase(main) :003 :0> create 'test', 'cf'
0 row(s) in 1.2200 seconds
hbase(main) :004 :0> put 'test', 'row1', 'cf :a', 'value1'
0 row(s) in 0.0560 seconds
hbase(main) :005 :0> put 'test', 'row2', 'cf :b', 'value2'
0 row(s) in 0.0370 seconds
hbase(main) :006 :0> put 'test', 'row3', 'cf :c', 'value3'
0 row(s) in 0.0450 seconds
```

- Get permits to obtain a single row.

```
hbase(main) :008 :0> get 'test', 'row1'
COLUMN CELL
cf :a timestamp=1288380727188, value=value1
1 row(s) in 0.0400 seconds
```

- By disabling and then dropping our table, we will clean up all done above.

```
hbase(main) :012 :0> disable 'test'
0 row(s) in 1.0930 seconds
hbase(main) :013 :0> drop 'test'
0 row(s) in 0.0770 seconds
```

- To exit the shell, we type exit.

hbase(main) :014 :0> exit

- We stop our HBase instance by running the stop script.

  $ ./bin/stop-hbase.sh
  stopping hbase.....

Hbase java API

HBase offers a java API permitting to get access to local and distant HBase and performing different operations.

We expose through the following example the basic API calls of HBase :

- We first create an instance of org.apache.hadoop.conf.Configuration. We ask the Configurationn class to create the instance. It will return a Configuration that has read HBase configuration from HBase configuration files found on the program's classpath :

  Configuration config = HBaseConfiguration.create() ;

- This Configuration is subsequently used to create instances of HBaseAdmin and HTable, two classes found in the org.apache.hadoop.hbase.client Java package. HBaseAdmin is used for administering your HBase cluster, for adding and dropping tables. HTable is used to access a specific table. The Configuration instance points these classes at the cluster the code is to work against. To create a table, we need to first create an instance of HBaseAdmin and then ask it to create the table named test with a single column family named data.

  HBaseAdmin admin = new HBaseAdmin(config) ;
  HTableDescriptor htd = new HTableDescriptor("test") ;

```
HColumnDescriptor hcd = new HColumnDescriptor("data");
htd.addFamily(hcd);
admin.createTable(htd);
```

- Operating on a table, we will need an instance of org.apache.hadoop.hbase.client.HTable passing it our configuration instance and the name of the table we want to operate on.

```
HTable table = new HTable(config, tablename);
```

- We can put a single cell value of value1 into a row named row1 on the column named data. The column name is specified in two parts : the column family name as bytes(databytes in the code above) and then the column family qualifier specified as Bytes.toBytes("1")).

```
byte [] row1 = Bytes.toBytes("row1");
Put p1 = new Put(row1);
byte [] databytes = Bytes.toBytes("data");
p1.add(databytes, Bytes.toBytes("1"), Bytes.toBytes("value1"));
table.put(p1);
```

- Next we create an org.apache.hadoop.hbase.client.Get, do a get of the just-added cell, and then use an org.apache.hadoop.hbase.client. Scan is used to scan over the table against the just-created table printing out what we find.

```
Get g = new Get(row1);
Result result = table.get(g);
System.out.println("Get : " + result);
```

- Finally, we clean up by first disabling the table and then deleting it. A table must be disabled before it can be dropped.

```
admin.disableTable(tablename);
admin.deleteTable(tablename);
```