Universidad de Valladolid

E. U. de Informática (Segovia)

Ingeniería Técnica en Informática de Gestión

ZATDROID

Satellite Tracking and Augmented Reality App for ANDROID

Alumno: Rodrigo Santos Álvarez

Tutores: Fernando Díaz Gómez
Jesús Álvarez Gómez

FINAL PROJECT

REPORT

ZatDroid

Satellite Tracking and Augmented Reality App for Android

1. ABSTRACT

The purpose of ZATDROID is to offer the user the possibility of tracking any artificial satellite. It locates it in Google Maps together with its predicted trajectory and also it allows the user to see it in real time in the sky with augmented reality camera view. The application is implemented in Java for ANDROID devices (tablets or smartphones).

Orbital mechanics calculations are developed following Newton equations and NORAD (*North American Aerospace Defence Command)* propagation models, firstly published in 1980 *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

ZATDROID downloads information from a data base of satellites provided by CELESTRAK.COM, does the orbital mechanics calculations (*NORAD models*), gets the device sensors magnitudes, connects to the web service `Google Maps Elevation` to get the altitude of the user location, processes data in terms of XML language, creates a GOOGLE MAPS views with the updated position in real time of the satellite picked and shows the position in the sky in augmented reality using OPENGL for the camera view.

The GUI (*Graphical User Interface*) manages to lead the users through an easy and friendly navigation to achieve their aims quickly, showing the results of the user picks with *BreadCrumbs*, supporting English and Spanish, showing icon-based menus and keeping the users informed of the longer processes by “*progress bars*”.

ABSTRACT

El objectivo de ZATDROID es ofrecer la posibilidad al usuario de encontrar satélites artificiales. Posiciona en GOOGLE MAPS junto con su trayectoria y permite al usuario verlo en tiempo real en el cielo con la vista de la cámara en realidad aumentada.

Los cálculos de mecánica orbital se han desarrollado siguiendo las ecuaciones de Newton y los modelos de propagación de NORAD (*North American Aerospace Defence Command),* publicados por primera vez en 1980 *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

ZATDROID descarga información de una base de datos de satélites ofrecida por CELESTRAK.COM, realiza los cálculos de mecánica orbital (modelos NORAD), obtiene las magnitudes de los sensores del dispositivo, se conecta al servicio web `Google Maps Elevation` para obtener la altitud de la localización del usuario, procesa datos en lenguaje XML, crea una vista de GOOGLE MAPS con la posición del satélite elegido actualizada en tiempo real y muestra la localización en el cielo en realidad aumentada usando OPENGL para la vista de cámara.

La GUI (*Interfaz gráfica de usuario*) gestiona la experiencia del usuario a través de una fácil y amigable interfaz de navegación para alcanzar los objetivos rápidamente, mostrando a los usuarios sus elecciones con un menú *BreadCrumbs*, ofreciendo todo en inglés y español, con menús basados en iconos y manteniendo a los usuarios informados de procesos largos con barras de proceso.

2. ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisors Fernando and Jesús for the continuous support of my project, for his patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me all the time.

Besides my advisors, I would like to thank my friend Rafael for all his expert advices, insightful comments and user experience knowledge.

Thanks also to all those anonymous people writing full and documented answers on the internet, without their daily help, I could not have achieved this Project.

My sincere thanks to "*travis*" from `mybringback.com,` author of the fantastic ANDROID video tutorials that introduced me into the Apps world, Maciej Grzegorczy, author of the SATFINDER app and T. S. Kelso webmaster of the fabulous `celestrak.com` for their generosity in spreading knowledge around the world.

Last but not the least, I would like to thank my family for supporting me spiritually throughout my life and being an example of effort and perseverance, especially Julia, who has suffered from lack of attention during the time dedicated to this project.

I wish this Project could help my future kid explore the world of physics and technology and wake up the sense of astonishment and surprise in Nature.

3. TABLE OF CONTENTS

1. Abstract 3
2. Acknowledgements 7
3. Table Of Contents 9
4. List Of Figures 11
5. Introduction 13
5.1. Identification 13
5.2. CD Struture 13
5.3. Objectives 13
5.4. State Of The Art 14
5.5. General Description 14
5.5.1. Functional Requirements 14
5.5.2. Screens Sequence Diagram 17
6. Orbital Mechanics 18
6.1. Coordinate Systems 18
6.2. Newton´S And Kepler´S Laws 19
6.3. Keplerian Elements 21
6.4. NORAD Orbit Propagation Models 22
6.5. Satellites Orbits 22
7. Breakthrough Design Features 24
7.1. Multilanguage Support 24
7.2. Multiple Screens Support: Icons, Text 24
7.3. Asynctask And Progress Bar: Running Code In Background. 24
7.4. Sharing And Storing Information 25
7.5. Breadcrumbs Navigation Buttons 26
7.6. GOOGLE Search Activity 27
7.7. Device Location Provider 27
7.8. Sensors Management 28
7.9. Smooth Movements Filter 28
7.10. Migration of SGP4 Code from FORTRAN / C To JAVA for ANDROID 28
7.11. Augmented Reality View: Layers Over Camera View 28
7.12. OPENGL Usage 29
7.13. GOOGLE MAPS View 29
7.14. XML/XSD Management 30
7.15. SAX Management 31
8. Planning and Budget 32
8.1. Work Estimation 32
8.2. Work Flow and Tasks 32
8.3. Schedule Estimated 35
8.4. Schedule Achieved 36
8.5. Costs Estimation 37
8.6. Detailed Budget 37
9. Future Work 40
9.1. Full Database Hosted In Server 40
9.2. Migration to ANDROID 4.3 New Features 40
9.3. Migration to IOS 40
10. List of References 41
11. List of Abbreviations 42
12. Appendices 43

4. LIST OF FIGURES

Figure 1. Description of the data flow and how the system manages to get the inputs, work with the data and export the two main outputs. 15
Figure 2. Explanation of the sequence of the screens through the application. 17
Figure 3. Orbital mechanics coordinate systems....18
Figure 4. Device coordinate systems 19
Figure 5. Orbits type, according to the eccentricity value 20
Figure 6. Keplerian elements diagram 21
Figure 7. Geostationary belt....22
Figure 8. Multilanguage screen....24
Figure 9. BreadCrumbs....26
Figure 10. GOOGLE Search & progress bar....27
Figure 11. Augmented reality view....29
Figure 12. GOOGLE MAPS View....30
Figure 13. Work Flow Stages 32
Figure 14. Estimated Schedule 35
Figure 15. Actual Work periods....36
Figure 16. Schedule achieved 36
Figure 17. Detailed Budget....37
Figure 18. Monetizing the App. Price Study....38
Figure 19. App Prices on the market....38
Figure 20. Downloads per App....39

5. INTRODUCTION

During the years I have been studying Computer Science, it has been always on my mind the idea of joining both of my study worlds: Space Engineering and Computers.

At the moment I had to decide the purpose of my final work project, I realized that it was the time to finally develop my initial idea. After a lot of work thinking and asking friends that are working in computer science and space, I came up with this project, ZATDROID. It includes skills from both sides of my career, keeping me motived and offering me the possibility to learn topics that I have never worked.

Moreover, I found that it could be a chance for many people interested in tracking artificial satellites from their devices: tablets and smartphones. Also this App brings closer to the non-expert user the satellites we are using every day whose identity is unknown for us.

The code has been developed in JAVA for ANDROID, using Eclipse with ANDROID SDK and has been tested and designed for a *Sony Xperia Neo V* with ANDROID 2.3. It has been tested in other devices with ANDROID 4 and works perfectly.

5.1. IDENTIFICATION

Título: ZATDROID: Satellite Tracking and Augmented Reality App for ANDROID
Autor: Rodrigo Santos Álvarez
Director: Fernando Díaz Gómez / Jesús Álvarez Gómez
Departamento: Computer Science (ATC, CCIA, LSI)

5.2. CD STRUTURE

CD:

- Documentation
  - Final_Year_Project_Report.pdf
  - Technical_Manual.pdf
  - User_Manual.pdf
- Software
  - Source Code
  - ZATDROID.APK
- Solapa.pdf
- Titulo.pdf
- Resumen.pdf

5.3. OBJECTIVES

1) To develop an ANDROID App, easy and friendly
2) To bring closer to the user artificial satellites orbiting the Earth
3) To be able to run the orbital mechanics calculations that predicts the position of a satellite.
4) To show the real time position of the satellite over GOOGLE MAPS, as well as its trajectory
5) To track the present position over the observer´s sky, using Augmented Reality.
6) To provide the app in English and Spanish.
7) To support different devices (models, trademarks, sizes)
8) To gain skills in ANDROID Software development and XML language.

5.4. STATE OF THE ART

On the one hand, the previous work consisting on searching information about existing applications, programs or studies has been important when starting to set the objectives and methodology. After some emails and internet search, this is a list of the applications that put together some of the functionalities that ZATDROID wants to implement:

- GPREDICT [1]
- VIS SAT [2]
- SATFINDER [3]
- SATELLITE AR [5]

On the other hand, the previous training in skills needed for the Project were to be considered. Most of the investment work was about the orbital mechanics part, which seemed to be much more specific tan computer science in ANDROID application development. Information about satellites georeference, device georeference, orbit prediction and methodology to solve mechanics equations has been difficult to find and understand. The main source is the document from NORAD (*North American Aerospace Defence Command*) which develops Perturbation models in orbits propagation *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

5.5. GENERAL DESCRIPTION

5.5.1. FUNCTIONAL REQUIREMENTS

INPUTS:

The starting point once you have decided to create an application giving information concerning satellites, is retrieving the sat description and main parameters of the orbit from a reliable and updatable source. Although there are sites where all magnitudes needed can be found (latitude, longitude…) directly just by developing a search inside the page, the objective of this project is more ambitious and the calculations are going to be done within the code. After searching the web for some time, CELESTRAK.COM finally fitted the requirements.

CELESTRAK.COM was created and is maintained by Dr. T.S. Kelso, recognized worldwide as an expert in the area of satellite tracking and orbit determination. Regularly sought out for advice and counsel by NASA, European Space Agency (ESA), Russian Space Institute, US and AF Space Command, and many universities and commercial space organizations.

The satellites data, together with their orbital parameters are to be found in a txt file called "TLEs (*two line element sets*)" from NORAD (*North American Aerospace Defence Command*) whose description is shown in appendix 1. Basically, it gives the orbital parameters in two lines with a known format. These parameters are measured in one moment in time every day, so that ZATDROID has an initial point to calculate the propagation for the orbit.

DATA MANAGEMENT:

<u>XML:</u>

Firstly, the txt file, in which TLEs are provided, are parsed into XML, following the example of appendix 2. It is also created a XSD (*XML Scheme Definition*) scheme.

Once the XML is ready, a search can be launched using SAX (*Simple API for XML*) in ANDROID to find the satellite picked and select all the data included in the XML file. Then a java object (sat class) is created with the information of the satellite picked.

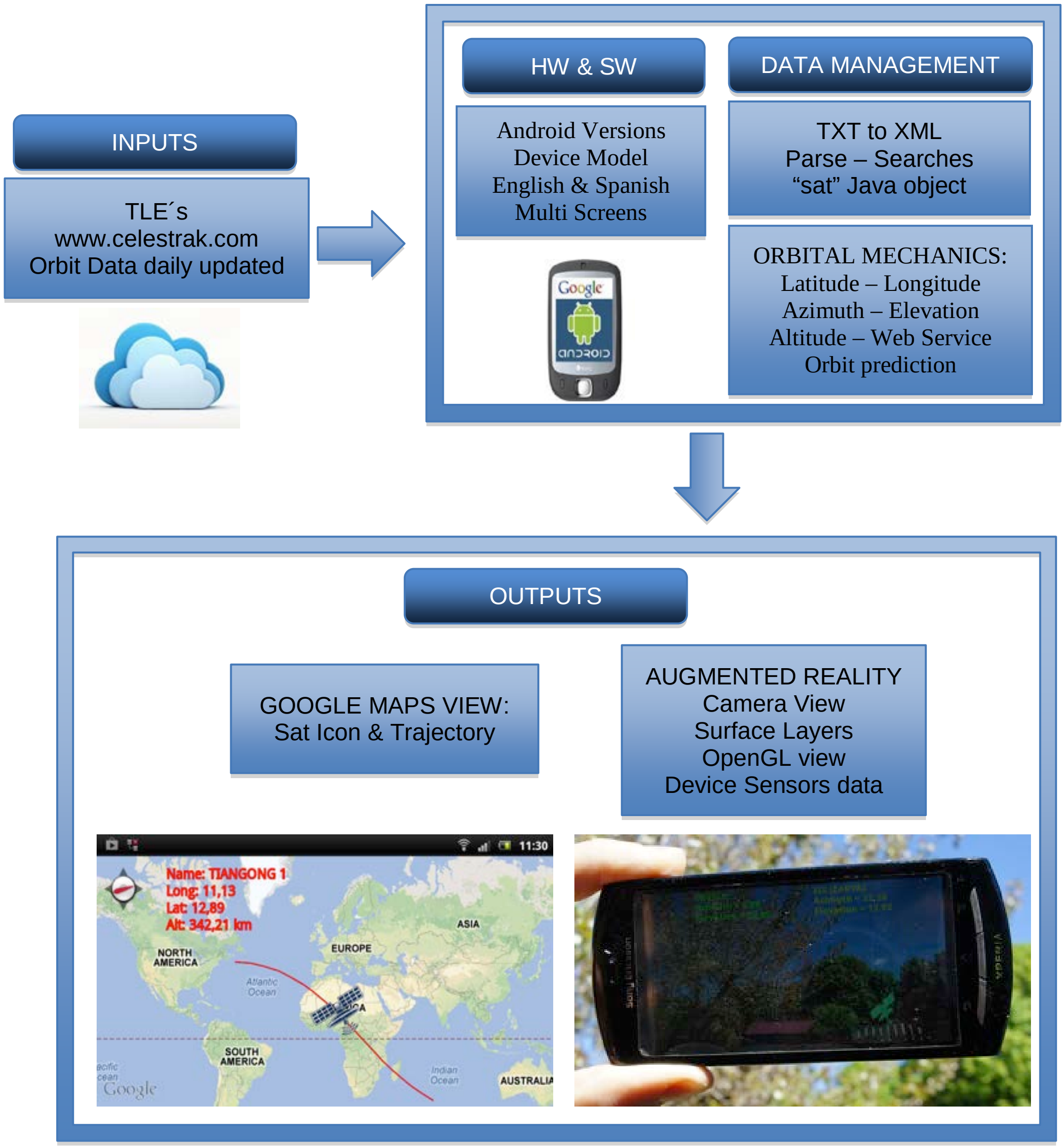


Figure 1. Description of the data flow and how the system manages to get the inputs, work with the data and export the two main outputs.

ORBITAL MECHANICS:

Longitude, latitude and altitude are magnitudes needed for the `Google Maps` positioning. Also azimuth and elevation of the satellite are essential for the augmented reality functionality.

The method to calculate these magnitudes was first introduced in 1980, *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4] where SGP (*Simplified general propagation model*) and SGP4 / SPD4 (*SGP version 4 for low and deep space orbits*) were implemented. These models develop the Newton´s Gravitational equations and some more accuracy modifications that takes into account perturbations. It was written in `FORTRAN`.

Some new developments have been achieved to improve the code using `C` language [6]. Nevertheless, the heart of the code remains the same. ZATDROID takes advantage of this implementation and develops all the calculations in Java for ANDROID. The orbital mechanics equations are explained in detail later on. ZatDroid implements SGP4/SDP4 NORAD prediction models, based on [4] and with some improvements of [6]

Furthermore, longitude, latitude and altitude of the users location is required for the augmented reality feature. GPS sensor provides these magnitudes. In case GPS is not available, network positioning is used for latitude and longitude and for the altitude, `Google Maps Elevation` web service is asked, though `USGS Elevation Query` is also another possibility.

OUTPUTS:

GOOGLE MAPS VIEW:

Once the application has all the parameters of the orbit and information of the satellite, it can be located inside a map view. The user is shown the latitude and longitude of the sat updated every second together with the name. In another layer, an icon is representing the satellite updated every second and finally the predicted trajectory for the next minutes.

AUGMENTED REALITY:

Taking advantage of a camera view, some layers are added to provide information concerning the position of the satellite in the sky where the user is. An icon must be drawn when the camera device is pointing at the azimuth and elevation of the satellite. Both azimuth and elevation of the satellite and the camera point of view need to known together with geographical coordinates (latitude and longitude) of the device.

Azimuth and elevation of the camera, that is to say, where is the camera pointing at, is retrieved from the accelerometer sensor. After some matrix transformations of the vectors from the coordinate reference systems of the device and the Earth, using OPENGL to set the camera view properly taking into account both directions of the satellite and the device, all the data is ready to draw the completed view.

While the user is rotating the device to find the satellite in the sky, information is being shown concerning the satellite and device parameters onto another layer. Also a line indicates the direction in which the user can find the satellite and finally the icon appear when the camera points at the right direction in the sky. The program gives information in case the satellite is not visible (under the horizon) and the time lasting until next overhead pass.

ZATDROID has the chance to implement an option that retrieves a list of satellites that are visible (over the horizon) at the moment where the user is located. It is not useful though, since it lasts some minutes to do all the calculations. For every satellite, SPG4/SPD4 model must be run, so the calculus time really increases.

HARDWARE AND SOFTWARE:

ZATDROID is required to:

- Run under ANDROID versions from 2.3 to 4.3 (*newest version up to dat*e).
- Support Multilanguage (*English and Spanish*)
- Support multiscreen sizes
- Support different smartphones models
- Tests have been carried out successfully with *Sony Xperia Neo V, Samsung Note, Samsung Galaxy S4, Samsung Tablet 11’, Nexus 4, Samsung S3-Mini*

5.5.2. SCREENS SEQUENCE DIAGRAM

A more complete explanation of the diagram can be found in other technical documents. The GUI (*Graphical User Interface*) is based on icons with explanatory text, taking advantage of ANDROID multilingual support: English and Spanish, depending on the language of the device and making the user experience unique and easy.

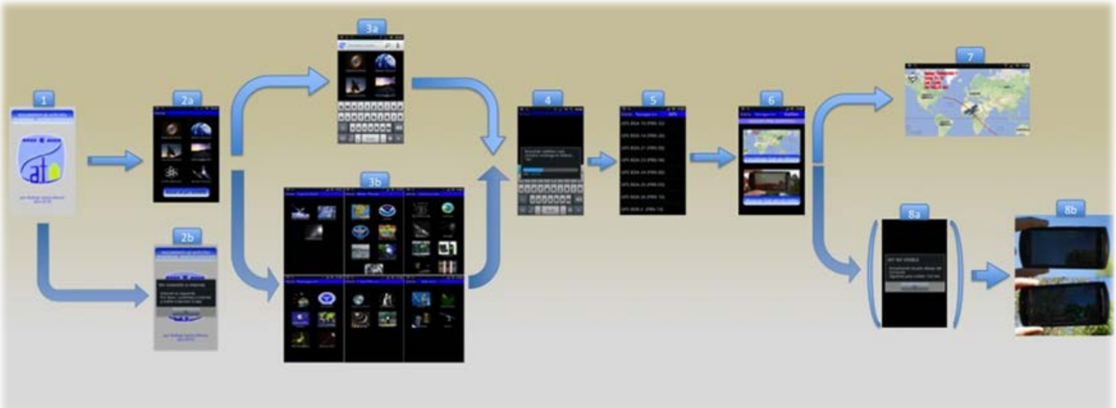


Figure 2. Explanation of the sequence of the screens through the application.

- **1-2b:** Firstly, the app checks if the device has internet connexion available. If not, a message pops up not to let the app execute. Internet is necessary.
- **2a**: Secondly, the user can choose between searching a satellite by its type or typing key words.
- **3a – 3b - 4**: Thirdly, after a progress bar to keep the user informed or some screens to decide the type, one specific satellite is finally picked. During this process, at the top of the screen the application implements *Bread Crumbs*. It allows the user to navigate through the types chosen.
- **6**: Once the satellite is picked, the screen shows the two functionalities of the application: `Google Maps` and Augmented reality with self-explanatory photos.
- **7**: For the `Google Maps` screen, it consists of the map view with two layers, indicating the latitude, longitude and name updated every second and the icon in the right placed also updated.
- **8a – 8b:** For the Augmented reality screen, in case the satellite is located under the horizon a message pops up with the time lasting until next overhead pass. Then the camera view is shown with some layers for the data of the satellite and data of the device updated every second, an arrow indicating the direction to follow in order to find the sat in the sky and the satellite icon when the device points at the right direction.

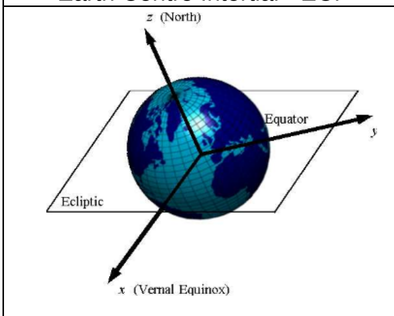
6. ORBITAL MECHANICS

This project makes sense since it involves a lot of knowledge in orbital mechanics. This section aims to be an introduction of the complex calculations used, as well as some general concepts concerning satellites. For a detailed description of the equations, processes and calculations, the technical manual is to be referenced.

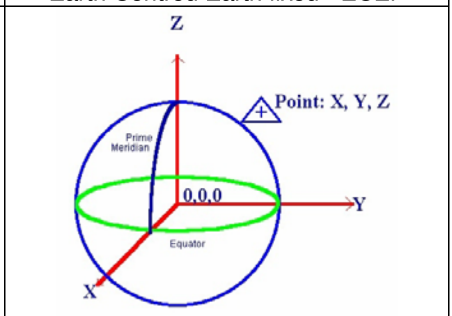
6.1. COORDINATE SYSTEMS

Orbital mechanics need some coordinate systems to develop all the calculations. Only a picture and the name is given below.

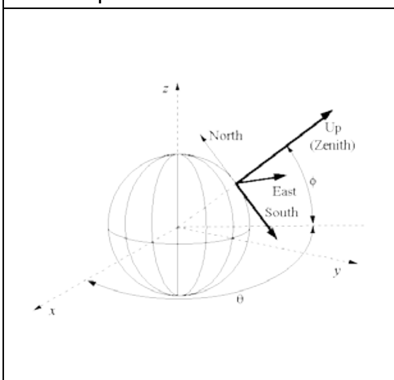
Earth Centre Intertial - ECI



Earth Centred Earth fixed - ECEF



Topocentric Horizon c.s.



Orbit c.s.

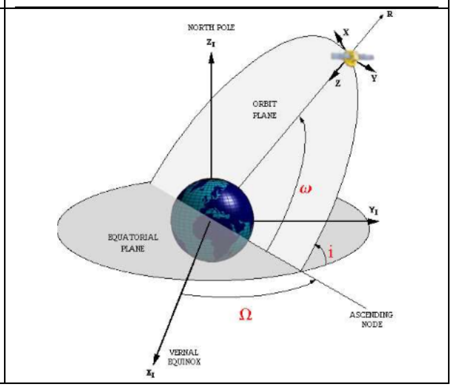


Figure 3. Orbital mechanics coordinate systems

An important coordinate system must be taken into account when working with devices such as smartphones or tables. Sensors (gravity, accelerometer and magnetic) give the magnitudes in terms of vectors, and every vector must be represented into a coordinate system, which is also used by OPENGL when working with 3D geometry or moving the camera in a camera view. The rotation matrix helps interpreting the vectors and their transformations.

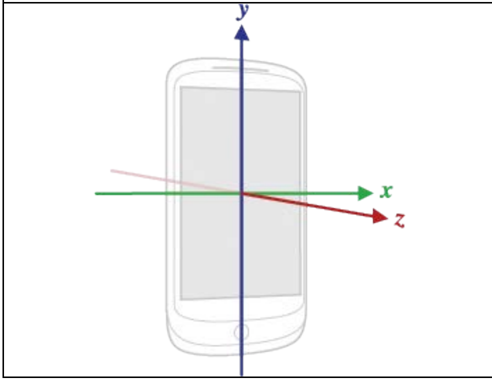
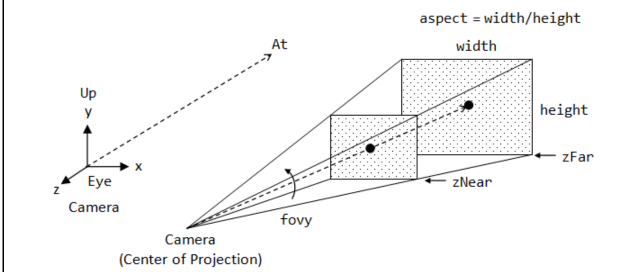
Sensors API c.s.	OPENGL Camera positioning
	

Figure 4. Device coordinate systems

6.2. NEWTON´S AND KEPLER´S LAWS

It was Kepler who at the beginning of the 17th century stated the 3 laws that describes the movement of the planets around the Sun. Kepler took advantage of the experimental work and measurements made by Tycho Brahe to realize these laws that can also be applied to the movement of any object in space around a more massive object:

- The orbit of every planet is an ellipse with the Sun at one of the two foci.
- A line joining a planet and the Sun sweeps out equal areas during equal intervals of time
- The square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit.

Later, Newton developed the fundamental laws of physics upon which the theory of orbital mechanics is based.

- **Newton's law of universal gravitation**, self explained by the well-known formula

$$\vec{F} = -\frac{GMm}{r^2}\frac{\vec{r}}{r} \qquad [7.1]$$

G = 6.67·10−11 Nm^2/kg^2 is the universal gravitational constant

The vector $\vec{r}$ goes from M to m and the force is on m.

- Newton's law of motion:

$$\vec{F} = m \cdot \ddot{r}\frac{\vec{r}}{r} \qquad [7.2]$$

Assuming that:

- The mass of a satellite is $m << M$ and $G(M+m) \approx GM$
- The gravitational forces are the only forces acting on the two bodies and these two bodies are not under influence of any other gravitational force from any other body.

And making use of Newton´s laws and some mathematical techniques, the three Kepler laws are proven in the next paragraphs:

1st LAW:
Solving the "*two bodies problem*" in polar coordinates and realizing the movement in such a central force is plain (the plane is determined by the position vector $\vec{r}$ and the velocity $\dot{\vec{r}}$) we can achieve the final formula that defines the orbits:

$$\vec{r} = \frac{\vec{L}_0^2 / GM}{1 + \frac{A}{GM}\cos(\nu)} = \frac{p}{1 + e\cos(\nu)} \quad [7.3]$$

$$\cos(\nu) = x \cdot \vec{r} / r \quad [7.4]$$

Depending of the value of e (eccentricity) the orbits will be:

- Ellipses: $0 < e < 1$
- Parabolas: $e = 1$
- Hyperbolas: $e > 1$

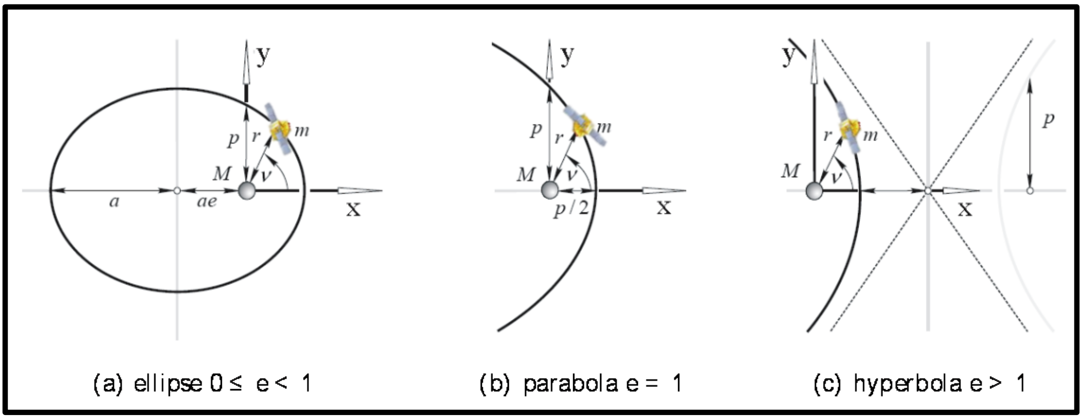


Figure 5. Orbits type, according to the eccentricity value [2]

2nd LAW:
Taking into consideration that the angular momentum of the motion is conserved since the gravitational force is a central force:

$$\vec{L} = \vec{r} \times (m \cdot \dot{\vec{r}}) = m(\vec{r} \times \dot{\vec{r}}) \quad [7.5]$$

$$\frac{\partial}{\partial t}\vec{L} = 0 \Rightarrow \vec{L} = \vec{L}_0 \quad [7.6]$$

$$\frac{dA}{dt} = \frac{1}{2} L_0 \quad [7.7]$$

It is proven that "A line joining a planet and the Sun sweeps out equal areas during equal intervals of time", meaning dA/dt the area swept out.

3rd LAW:

$$\frac{dA}{dt}=\frac{A}{T}=\frac{\pi a^2\sqrt{1-e^2}}{T} \quad [7.8]$$

$$\frac{dA}{dt}=\frac{1}{2}L_0=\frac{1}{2}\sqrt{GMa\left(1-e^2\right)} \quad [7.9]$$

$$T^2=\frac{4\pi^2}{GM}a^3 \quad [7.10]$$

a: semimajor axis of the orbit.

Final result for T (period) equation comes from [7.8] and [7.9] calculating the same magnitude through two different ways.

6.3. KEPLERIAN ELEMENTS

With all the data retrieved from Newton´s laws it can be stated for ellipse orbits (the most common ones for satellites) that there are six constants that describes easier the orbit than using the equations. These so called keplerian elements are not constant if disturbance forces are considered and all of them become time-variant. (*see next section*)

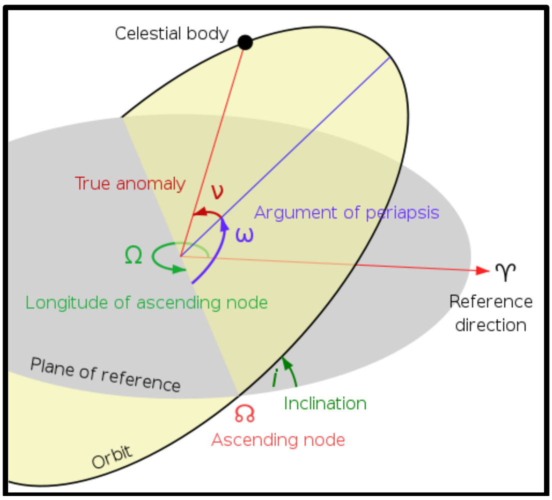


Figure 6. Keplerian elements diagram

- Shape and size of the ellipse.
  1) **Eccentricity (*e*):** already explained.
  2) **Semimajor axis: (*a*):** the sum of the periapsis and apoapsis distances divided by two.
- Orientation of the orbital plane in which the ellipse is embedded:
  3) **Inclination (*i*)** vertical tilt of the ellipse with respect to the reference plane, measured at the ascending node
  4) **Longitude of the ascending node (Ω):** horizontally orients the ascending node of the ellipse with respect to the reference frame's vernal point

- Finally:
  5) Argument of periapsis (ω): defines the orientation of the ellipse in the orbital plane, as an angle measured from the ascending node to the periapsis.
  6) Mean anomaly at epoch (M_0): defines the position of the orbiting body along the ellipse at a specific time (the "*epoch*").

6.4. NORAD ORBIT PROPAGATION MODELS [2]

Disturbance forces can be minute in comparison to the main gravitational force. Nevertheless, if we are to calculate a more accurate prediction, these forces must be taken into account.

The North American Aerospace Defence Command (NORAD) developed some Perturbation models in *SpaceTrack* 3 [4], depending on its accuracy:

- **SGP** (*Simplified General Perturbation model*):
  - Geopotential disturbances: the Earth is not spherical. It has a bulge at the equator and is flattened at the poles.
  - Perturbances due to Sun and the Moon: more tan only one object in space causes perturbations.
- **SGP4** (*Simplified General Perturbation model version 4*) adding:
  - Atmospheric drag: removes energy from the satellite.
  - Solar radiation: less important in lower orbits.
- **SDP4** (*Simplified General Deep Space Perturbation model version 4*)
  - It is based on SGP4 but for orbits with a period longer than 225 minutes, where additional deep-space perturbations have to be considered.

ZatDroid implements SGP4/SDP4 perturbation models described in [4] and with some improvements of [6]. Code from C, FORTRAN or PASCAL has been migrated to JAVA for ANDROID

NORAD maintains a set of elements needed to do the calculations for the propagation of the orbits. These elements are refined every day so that the model keeps its accuracy and can be found as txt files called TLEs (*two line element sets).* CELESTRAK.COM is the source from ZATDROID downloads these elements every time the user executes the app.

6.5. SATELLITES ORBITS

Finally, a brief description of the orbits is explained, as it is written in [8]

SYNCHRONOUS:

<u>EARTH SYNC</u>.

Also known as Geostationary. It allows the satellite to look always at the same point of the Earth. Therefore, it is highly demanded and there is a belt with a lot of satellites. Its altitude is roughly 35.876Km, flying at 3Km/s over the equator, inclination 0.

In ZATDROID these satellites can be recognized easily as their latitude and longitude are constant and no trajectory line is drawn in the `Google Maps` view.

Figure 7. Geostationary belt

SUN SYNC:
These orbits allows a satellite to pass a section of the Earth at the same time of the day. The surface illumination angle will be nearly the same every time. This consistent lighting is a useful characteristic for satellites that image the Earth's surface in visible or infrared wavelengths (e.g. weather and spy satellites) and for other remote sensing satellites (e.g. those carrying ocean and atmospheric remote sensing instruments that require sunlight)

POLAR:
The satellite passes over the planet's poles on each revolution. The inclination of these orbits therefore is $i \approx 90º$. These orbits are the only ones that allows to pass the poles to search any information.

7. BREAKTHROUGH DESIGN FEATURES

7.1. MULTILANGUAGE SUPPORT

Following ANDROID developers advices, all strings used in ZATDROID have been encoded inside the XML file `strings.xml` in English, which is the main language. This file is located in *res/values* There is also a file `string-es.xml` in *res/values-es* in which all strings are duplicated Spanish.

Whenever the user runs ZATDROID, the language of the strings are shown in the language of the device automatically, as long as it is English or Spanish. English is by default.

Figure 8. Multilanguage screen

7.2. MULTIPLE SCREENS SUPPORT: ICONS, TEXT

Bearing in mind the wide range of devices it is mandatory to support different screen sizes. Although in the ANDROID help it is recommended the use of “*dp*” (*density pixels*) when defining pictures or text size in XML layouts, it has not been the solution this time.

Many tests have been run with different devices (*Sony Xperia Neo V, Samsung Note, Samsung Galaxy S4, Samsung Tablet 11’, Nexus 4, Samsung S3Mini*) and using “*dp*” did not show the icons and text the same way in all devices. So another definition of the layouts was necessary.

Finally, it was found out that using only xml layouts was not the right procedure. Instead, layouts definition inside Java code was selected. This option lets the programmer customize deeper the layout configuration. `addView` adds views (layouts) to other layouts.

- On the one hand, icons needs to set height and width according to the size of the device. This is achieved by retrieving the device width and height with this piece of code:
```java
final float height=getResources().getDisplayMetrics().heightPixels
final float width=getResources().getDisplayMetrics().widthPixels
```
  Every icon height and width can be configured as a certain percentage of total:
```java
int h = (int) (0.05 * height); // 5%
int w = (int) (0.20 * width);  // 20%
```
- On the other hand, text size (`setTextSize`) and gap between letters (`setTextScaleX`) is something more complex. The text must adjust to the size of its box perfectly.

Gap between letters:

A new function has been created `scaleButtonText`.

- o A `Paint` object is created with 100% height and setTextScaleX = 1
- o Ask the `paint` for the bounding rectangle if it were to draw this text.
- o Determine the width
- o Calculate the new scale in x direction to fit the text to the button width

Text Size:

It is set to a percentage of the total height of the box.

7.3. ASYNCTASK AND PROGRESS BAR: RUNNING CODE IN BACKGROUND.

There are two events within the application when a connection through internet is established to download files. This happens when connecting to CELESTRAK.COM and downloading TLEs in a txt file. Such a process may last some seconds and give errors. It is also the moment when both XML and XSD file is created from txt.

`Asynctask` is defined in the ANDROID developers site:
"This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.
An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called `Params`, `Progress` *and* `Result`, *and 4 steps, called* `onPreExecute`, `doInBackground`, `onProgressUpdate` *and* `onPostExecute`"

So this every time ZATDROID downloads a file from the internet, `asynctask` is launched and a progress bar is shown to the user to keep informed of the task that are being performed.

7.4. SHARING AND STORING INFORMATION

ZATDROID requires to share information: files, java objects, parameters. There are many ways to share information in ANDROID.

SQLITE DATABASES:

In case a complex data management is needed with all the advantages of a relational data base. ZATDROID makes no use of this option because there is no need to manage a lot of information with tables and relationships. Also, XML files has been picked as the way to manage data because of its simplicity and the objective to learn concerning XML during this project.

PASSING OBJECTS: [9]

- SERIALIZABLE Class:

The problem with this approach is that reflection is used and it is a slow process. This mechanism also tends to create a lot of temporary objects and cause quite a bit of garbage collection.

- PARCELABLE Class

This code will run significantly faster. One of the reasons for this is that we are being explicit about the serialization process instead of using reflection to infer it. It also stands to reason that the code has been heavily optimized for this purpose.

To pass the PARCELABLE object between activities, `Intent.putExtra` is used.

ZATDROID uses the "sat" class that implements PARCELABLE and passed between activities providing the information of the satellite picked.

PASSING PARAMETERS:

- SHAREDPREFERENCES [10]

"The `sharedPreferences` *class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use* `sharedPreferences` *to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed)."*

ZATDROID uses `sharedPreferences` to share Strings that save names of satellites picked for the user.

- <u>Final Static</u>
  Constants are storaged in classes this way.

ZATDROID has a class named *k* for all constants used in mathematical calculations apart from other constants in other classes.

SHARING FILES:

- <u>External - Internal Storage</u>: [11]
  When building an app that uses the internal storage, the Android OS creates a unique folder, which will only be accessible from the app, so no other app, or even the user, can see what's in the folder.

The external storage is more like a public storage, so for now, it's the SD card, but could become any other type of storage (remote hard drive, or anything else).

The internal storage should only be used for application data, (preferences files and settings, sound or image media for the app to work). The external storage is often bigger. Besides, storing data on the internal storage may prevent the user to install other applications.

ZATDROID uses internal Storage to save the XML and XSD files because they are small and are used only for the application.

7.5. BREADCRUMBS NAVIGATION BUTTONS

In every screen at the top the user can find some buttons as indicated in figure 8. These so-called `BreadCrumbs` functions are:

- To be helpful as they provide information of the satellite type picked during the process of several screens. All types are shown so that the user, at first sight, remembers the types and name of the satellite picked.
- To allow the user to navigate through the previous screens, changing the type picked, just by clicking on one of the buttons of the breadcrumbs.

The layouts affected are created programmatically, not with XML. This is another example where layouts defined with code allow to add some more functionalities than with XML.

Figure 9. BreadCrumbs

7.6. GOOGLE SEARCH ACTIVITY

The first screen gives the opportunity to select searching a satellite by it type or launching a search by some key words. See Figure 9.

This last functionality uses GOOGLE Search Activity from Android. The button calls `onSearchRequested` method and the GOOGLE interface is launched. It opens a dialog with a keyboard where you can write key words to find a satellite. It also supports voice search.

CELESTRAK provides txt files with TLEs classified by type. To be able to perform the search in all satellite files provided by CELESTRAK, ZATDROID needs to download all files, merge them together in one file and then search. This process lasts some seconds, it is encoded inside an `Asynctask` and a progress bar is shown to the user.

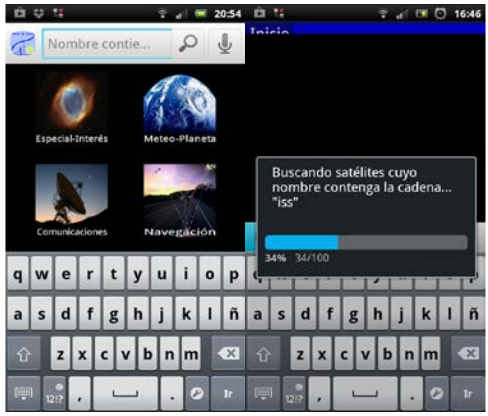


Figure 10. GOOGLE Search & progress bar

7.7. DEVICE LOCATION PROVIDER

Latitude, longitude and altitude of the device must be known whenever a prediction for the location of the satellite in the sky is intended to do. Besides knowing the satellite georeference, also device coordinates are essential.

With the help of [12] ZATDROID implements a customized `LocationListener` to retrieve **latitude** and **longitude** coordinates:

- Checks that Network or GPS is enabled. If not, it shows a message to the user requiring at least one connection.
- Firstly retrieves location from Network provider. It is quicker and usually devices enables network but not always GPS.
- Secondly, if GPS is enabled, as it is more precise, uses it to get a more accurate location.

Finally, **altitude** is also required.

- If GPS is enabled, it is easy. GPS gives it together with latitude and longitude.
- Otherwise, a web service is called using a `httpRequest`. There are two possibilities based on 3D maps of the Earth: [13]
  - USGS Elevation Query Web Service: [14]
  - Google Maps Elevation API Web Service: [15]

ZATDROID implements `Google Maps Elevation API Web Service`, although there is no big difference between them, advantages or disadvantages.

7.8. SENSORS MANAGEMENT

Method `mSensorManager` provides access to all sensors installed in the device. Three sensors are to be used by ZatDroid:

- **Accelerometer**: measures the acceleration applied to the device that gives the orientation.
- **Magnetic Field**: magnetic vector is provided device coordinate system.
- **Gravity**: Gravity vector is given in device coordinate system.

Vectors need a reference coordinate system and as the device is moving and rotating, the transformation matrix is also necessary. Two methods inside `mSensorManager` (`getRotationMatrix` and `getOrientation`) give information about the orientation and coordinate system of the device and the transformation matrix.

7.9. SMOOTH MOVEMENTS FILTER [16]

The augmented reality functionality requires to retrieve device orientation in real time. This information is used to locate the satellite icon on the screen when the user is rotating the device searching the right orientation (azimuth and elevation). As a result, the icon is moving through the screen.

This movement depends on the accuracy, updating time and speed of rotating the device. One method has been implemented to smooth the movement of the icon on the screen: `filterSmoothMovements`. It takes two factors into account:

- `SmoothFactorCompass`: so that the small jumps do not disturb
- `SmoothThresholdCompass`: minimum distance so that the icon jumps

7.10. MIGRATION OF SGP4 CODE FROM FORTRAN / C TO JAVA FOR ANDROID

Document [4] stated the definition of the methods to calculate the propagation of the orbits taking into account perturbations due to disturbance forces. This was in 1980 in FORTRAN language. A new revision was developed in 2006 in C [6] Some other reviews and improvements have been implemented but the heart still remains.

ZATDROID implements the code from 1980 [4] with some modifications (not all) from 2006 [6] migrated to JAVA for ANDROID.

7.11. AUGMENTED REALITY VIEW: LAYERS OVER CAMERA VIEW.

Augmented Reality views means to be able to merge the camera view with some other artificial layers with digital information and let both worlds interchange information and interact with each other.

This has been achieved thanks to the `GLSurfaceView` and `SurfaceView` classes and then overlapping them with `addViews`. All these layouts have been created programmatically, as already explained before to increase customization.

ZATDROID implements 6 layers at the same time: (Figure 10)

1) The camera view, as usual.
2) A rectangle in the centre that changes colour when device orientation is directly pointing at the satellite in the sky
3) An arrow indicating the direction of the satellite to rotate the device
4) An icon symbolizing the satellite in the right azimuth and elevation.
5) `TextView` with the name and orientation (azimuth and elevation) of the satellite
6) `TextView` with elevation and azimuth of the device updated in real time.

Figure 11. Augmented reality view

7.12. OPENGL USAGE

When implemented AR View, it was complicated to visualize the icon of the sat only when the device was pointing at the direction of the satellite. OPENGL has been used to rotate the virtual camera that shows the icon at the same time as the device camera is rotated by the user.

- `GLU.gluLookAt` and `GL10.glFrustumf` control the focus, position and movements of the virtual camera.
- As explained before, the moving coordinate systems made the vectors transformations really complicated. Complex matrixes were used.
- The geometry of the arrow and rectangle was easy, but the icon was inside a rectangle and was always moving, so updating the geometry of the icon really increased the difficulty.

7.13. GOOGLE MAPS VIEW

As one of the two main functionalities, locating the satellite icon in the GOOGLE MAPS in real time with latitude and longitude is exciting. These maps have the same appearance as the maps you see in your computer, but here you can customize to show whatever you want to (Figure 10)

- `MapActivity` is the object used in ZATDROID. Newer versions of Android has deprecated `MapView` and replaced it with `GoogleMap` object. The heart is similar but other complements are improved. For this app, `MapView` fits all requirements.
- `com.google.android.maps.Overlay` is the layer where all the information is added and shown onto the map.

- A canvas let the program add the satellite icon as a bitmap together with the `geoPoint` represented by the latitude and longitude of the satellite.
- Also in this canvas a `drawPath` object allows to draw a trajectory line. This trajectory represents the orbit in its last 20 minutes and its predicted 20 coming minutes, all calculated iterating the propagating models.
- And the last layer is the `textView` with name, latitude and longitude of the satellite updated in real time.

The satellite icon is updated almost instantly. The refreshing delay for the trajectory is one second though, that is, the trajectory stays old when zooming or moving the map. It was tested to update the trajectory several times per second so that the visual appearance was smoother, but the high amount of calculations, made it not possible for the program.

If the user ever sees a satellite at high altitudes (30.000 Km) and the trajectory is not drawn, it is not a bug. The satellite follows a geostationary orbit and therefore it is fixed. Actually, it is rotating with the Earth, so for an inhabitant it is always in the same position, same latitude and longitude.

Figure 12. GOOGLE MAPS View

7.14. XML/XSD MANAGEMENT

The TLEs downloaded from CELESTRAK is provided in txt format as described in appendix 1. One of the objectives of this project was to learn XML language. Therefore, the data management in ZATDROID was in XML. So TLEs are transformed into XML. The XML format is shown in appendix 2.

The way to create XML from the txt is done manually, by parsing txt NORAD format and then writing XML tags and content. The files are storaged in the internal memory, as explained before in this section.

7.15. SAX MANAGEMENT

SAX (*Simple API for XML*) and DOM (*Document Object Model*) are the two methods to deal with XML data in ANDROID. A brief description is now given: [17]

SAX:

- Parses node by node
- Doesn’t store the XML in memory
- We can´t insert or delete a node
- SAX is an event based parser
- SAX is a Simple API for XML
- Doesn’t preserve comments
- SAX generally runs a little faster than DOM

DOM:

- Stores the entire XML document into memory before processing
- Occupies more memory
- We can insert or delete nodes
- Traverse in any direction.
- DOM is a tree model parser
- Document Object Model (DOM) API
- Preserves comments

ZATDROID is:

- Dealing with big documents
- Not inserting nodes, just reading
- Intending not to use a lot of memory
- Not reading the whole document, just the node required.
- Not needing to store the whole document and creating the DOM tree.

So SAX was decided to be the method for the searches into XML. ZATDROID looks into the XML for the satellite picked by the user and read all the information of this certain satellite to create afterwards a “sat” JAVA object.

8. PLANNING AND BUDGET

8.1. WORK ESTIMATION

FUNCTION POINTS is considered a useful tool to make an estimation of the lines of code. But this project has some special characteristics that may not fit with a typical estimation, so that the results would not be coherent.

- Only one person is in charge of the whole project
- The training stage must be extremely large, as the programmer is untrained.
- The orbital mechanics stage is out of the reach of the FUNCTION POINTS analysis
- There is no routine in the time dedicated every week to the project because the programmer has a full time job. The project is being developed in his free time.
- In any moment the project could be delayed due to any reason.

So the planning is created from the stages and taking into account experts advices, self experience in other projects from other disciplines.

8.2. WORK FLOW AND TASKS

The work flow defines the stages of the project. Figure 12.

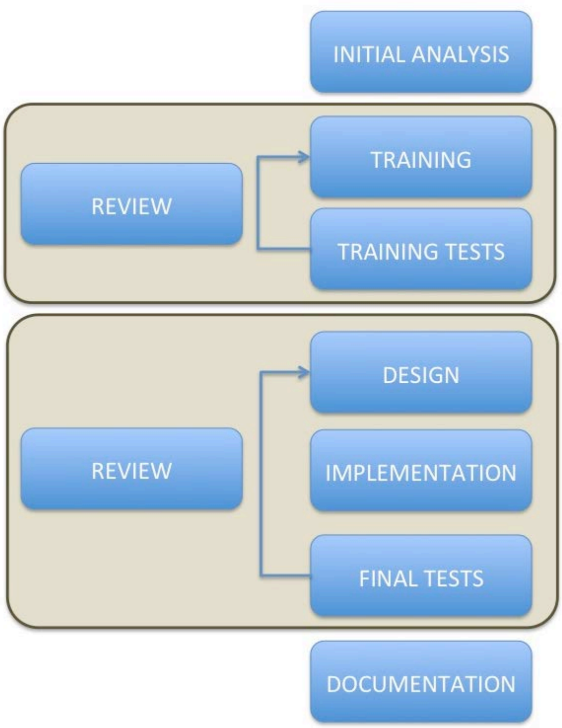


Figure 13. Work Flow Stages

Taking the stages definition as a starting point, a detailed tasks breakdown structure can be made.

1. **Initial analysis**
    1.1. Objectives definition
    1.2. Resources analysis
        1.2.1. Human Resources
        1.2.2. Technical resources
    1.3. Schedule
        1.3.1. Initial tasks definition
        1.3.2. Initial requirements definition
        1.3.3. Schedule estimated
2. **Training**
    2.1. Android programming. SDK
        2.1.1. Eclipse SDK
        2.1.2. Activities flow
        2.1.3. Layouts
            2.1.3.1. Types
            2.1.3.2. Components
            2.1.3.3. Programmatically
            2.1.3.4. Threads
        2.1.4. Parameters
        2.1.5. Classes
        2.1.6. Sharing and passing information
    2.2. Google Maps View
        2.2.1. MapView
        2.2.2. Layers and Markers
        2.2.3. Geometry
    2.3. Augmented reality views
        2.3.1. Camera View
        2.3.2. Layers
        2.3.3. Geometry
    2.4. OpenGL
        2.4.1. Coordinate systems
        2.4.2. Transformation Matrix
        2.4.3. Geometry
    2.5. XML management
        2.5.1. Understanding the language
        2.5.2. Creating files
        2.5.3. XSD scheme
        2.5.4. Search: SAX, DOM
    2.6. Orbital Mechanics calculations. TLEs. SGP4/SDP4.
        2.6.1. Satellites orbits
        2.6.2. Equations
        2.6.3. Methods understanding
        2.6.4. Programming code
    2.7. State of the art: applications
3. **Training Tests** (carried out at the same time as the training)
    3.1. Activities tests
    3.2. Sensor tests
    3.3. OpenGL tests

3.4. Google Maps tests
3.5. XML Management
3.6. Augmented Reality tests: camera view and layers
3.7. Orbital Mechanics Calculations tests.

4. Design
4.1. Requirements.
4.1.1. Analysis of the estimated requirements
4.1.2. Set the final requirements
4.2. Classes diagram
4.2.1. Activities classes
4.2.2. Calculations Classes
4.3. Sequence Diagram
4.4. GUI Design

5. Implementation
5.1. Android core
5.1.1. Activities Structure
5.1.2. Sensors
5.1.3. Sharing and passing data
5.2. GUI
5.2.1. Screen definition
5.2.2. Icons and Text
5.2.3. Multilanguage support
5.3. XML files management
5.3.1. XML Parsing
5.3.2. XSD Creation
5.3.3. Search: SAX Handling
5.4. Google Maps Activity
5.4.1. MapView
5.4.2. Markers
5.4.3. Geometry
5.4.4. Layers
5.5. Augmented Reality
5.5.1. Camera View
5.5.2. Layers
5.5.3. OpenGL Geometry
5.5.4. Device orientation sync with sat icon
5.6. Orbital Mechanics Code. SGP4/SDP4
5.6.1. TLE download
5.6.2. Time calculations
5.6.3. SGP4/SDP4 Code

6. Final Tests
6.1. Activities structure
6.2. GUI
6.2.1. User experience
6.2.2. Activities
6.3. AR functionality
6.3.1. Layers
6.3.2. Icon movement with device orientation
6.4. GOOGLE MAPS functionality
6.4.1. Icon marker

6.4.2. Trajectory line
6.5. Orbits propagation check
6.6. Devices compatibility
6.7. Android versions compatibility

7. Documentation

7.1. Final Year Project Report
7.2. Technical Manual
7.3. User Manual

8.3. SCHEDULE ESTIMATED

Not all the tasks are written in the open project file, as they are too specific to give a general view.

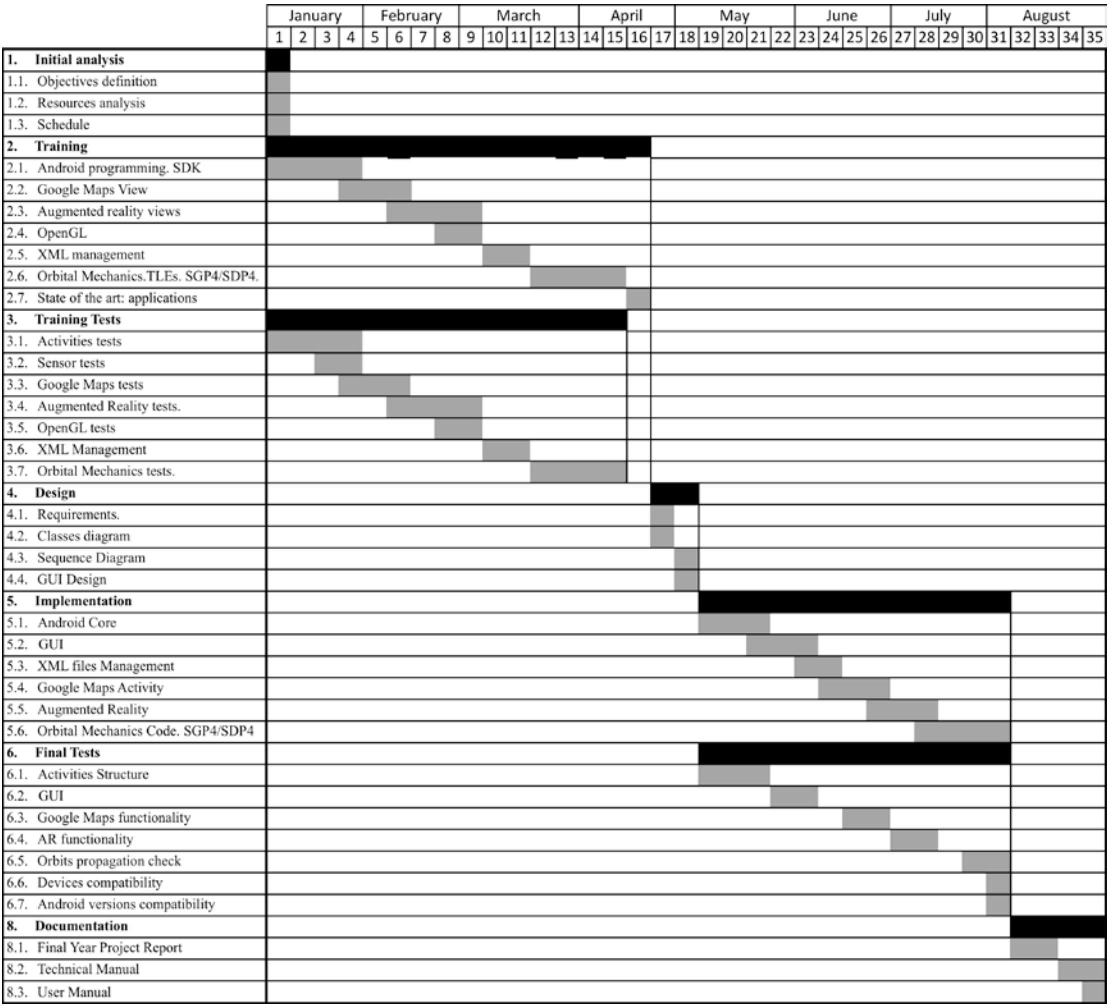


Figure 14. Estimated Schedule

The total estimated time is 35 weeks, with one person working full time in the project, 5 days a week, 1400 hours.

8.4. SCHEDULE ACHIEVED

The beginning of 2012 was the start point of this project. After some months working, I was offered a new job, so from July 2012 until December 2012 the project stayed in stand-by. Then, in January 2013 it all begun again and in May 2013 I finally finished it. June was again dedicated to my job, and July and August has been the time to finish. So I had 13 months of actual work in three periods:

January - June 2012 January	January - May 2013	July-August 2013
6 Months	5 Months	2 Months
13 Months		

Figure 15. Actual Work periods

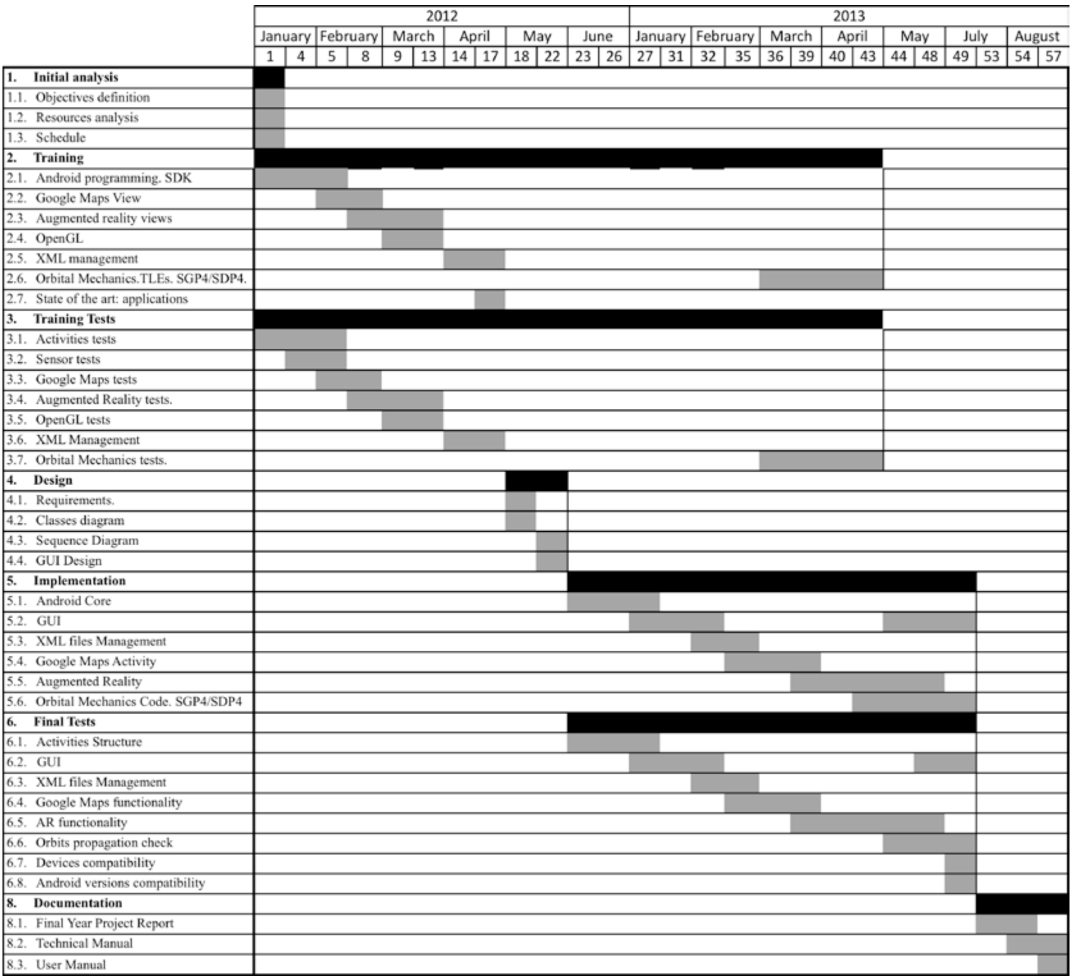


Figure 16. Schedule achieved

Moreover, only 4 hours per day (at most) were available to go forward with the project, although weekends were also dedicated to the project. So, from the 8 months estimated, one person full time, the project has been developed in 13 months. It may be thought that 16 (double) months were needed, but if holidays and weekends are to be added to the schedule, 13 months are enough. It may be agreed that 1400 hours is a good estimation for the project.

The main deviation noticed about technical issues has been that, after the first implementation of the GUI with `ListViews`, tests performed with users revealed that the GUI was not friendly and gave no information to the user about choices made in the satellite type. So a new process of improving the GUI was carried out:

- Adding `BreadCrumbs` to the screens
- Changing `ListViews` for icon based layouts.

After June 2012, the implementation was started and most of the training finished. So the implementation in 2013 went forward quickly. The key point that was entirely performed in 2013 was the Orbital mechanics training and implementation. Actually, training and implementation was mixed and this feature was the last to be finished.

8.5. COSTS ESTIMATION

Once again, there are some tools to estimate costs for a project. COCOMO may be one of the most used in the subjects of the degree. But for this case, no tool has been used to calculate an estimation, because the project is simple looking from the costs point of view.

- Only one person is working on it.
- Most of the time has been invested in training. Are these hours costs?
- The material resources are easily measured.

A final budget is detailed in the next section, which would be completely similar to the estimated one, except for little things.

8.6. DETAILED BUDGET

Resource	% Use	Total Resource Cost	Total cost for the project
Computer	50%	1000€	500€
MS Office 2010 Prof	25%	480€	120€
Eclipse SW	100%	0	0
Start UML	100%	0	0
Printer & toner	10%	200€	20€
Paper office material	100%	20€	20€
ADSL (13 months)	25%	400€	100€
Sony Ericson Neo V	75%	150€	112€
5 project copies printed and bookbound	100%	300€	300€
		TOTAL:	1.172€

Human resources	€ per Hours	Hours	Total cost for the project
Computer Science Engineer	12€	1.400€	16.800€
		TOTAL:	16.800€

Figure 17. Detailed Budget

So the final costs are **17.972€.** This is a huge quantity for the project.

- Half of the working hours have been training. These may be not taken into account as the programmer is supposed to have the skills before the contract. So the prize would reduce 8.000 €, and the app final cost would be **9000€.**
- Bearing in mind the idea of **monetizing** the App in the market, if we sell it in Google Play:
    - Costs are 25€ once to register.
    - The App. price: 70% for the seller, 30% for Google.

MONETIZING THE APP		
Objective: 9.000€		
App Price	€ for the author	Downloads
€ 0,20	€ 0,14	64.286
€ 0,30	€ 0,21	42.857
€ 0,40	€ 0,28	32.143
€ 0,50	€ 0,35	25.714
€ 0,60	€ 0,42	21.429
€ 0,70	€ 0,49	18.367
€ 0,80	€ 0,56	16.071
€ 0,90	€ 0,63	14.286
€ 1,00	€ 0,70	12.857
€ 1,10	€ 0,77	11.688
€ 1,20	€ 0,84	10.714
€ 1,30	€ 0,91	9.890
€ 1,40	€ 0,98	9.184
€ 1,50	€ 1,05	8.571

Figure 18. Monetizing the App. Price Study.

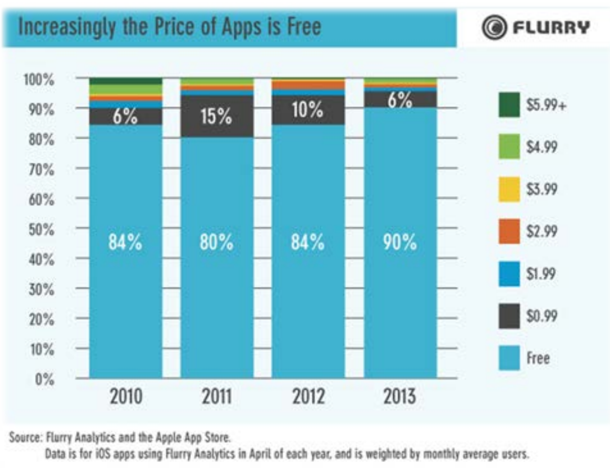


Figure 19. App Prices on the market

Figure 18 explains how the rate of users downloading free apps is increasing.

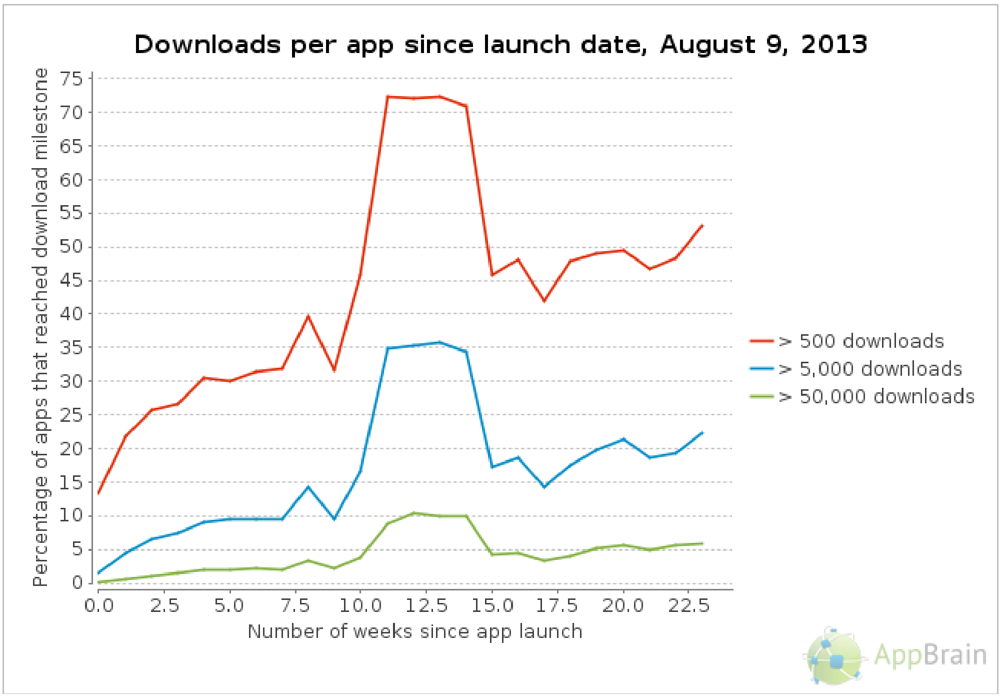


Figure 20. Downloads per App

Figure 19 shows how difficult is to gain more than 5.000 downloads. Only 20% (average) of the apps reach more than 5.000 downloads after 20 weeks in the market.

With the comparison of the price and downloads needed to reach 9.000€, it is difficult for ZATDROID to be profitable.

9. FUTURE WORK

Once the initial objectives are achieved and after developing this project, there are some issues that could be improved or added to the requirements of ZATDROID. Here are some of them:

9.1. FULL DATABASE HOSTED IN SERVER

It would be nice to the user experience that there is no need to choose a satellite to performed a GOOGLE MAPS View or an Augmented Reality View. Instead, a complete database with all satellites from CELESTRAK could be developed. Every satellite would have the real time parameters updated so that the device could show all of them (or just some with filters) when initializing the Google Maps View or the Augmented reality view.

This new feature would require a large computational. Therefore, it would mean a change in the core of the project. The idea should be to implement the calculation modules in a server and let the device connect to the server to provide a wide range of options in the visualizations functionalities.

9.2. MIGRATION TO ANDROID 4.3 NEW FEATURES

ZATDROID has been developed under ANDROID 2.3. and tested in a *Sony Xperia Neo V*. From the start of the project (January 2012) ANDROID has launched new versions with new features: OPENGL 3.0, Optimized Location and Sensor Capabilities, transparent overlays, fragments, Action Bar…

9.3. MIGRATION TO IOS

It would be interesting to be able to migrate the whole app to IOS, as it will be an opportunity to expand the market. Nevertheless, this is complicate as it involves lot of modifications.

10. LIST OF REFERENCES

[1] Csete, A. (2009). GPREDICT: Free, Real-Time Satellite Tracking and Orbit Prediction Softtware. Retrieved from http://gpredict.oz9aec.net/

[2] Daum, P. (2005). VIS SAT: A Satellite Footprint Visualization Tool (Master Thesis). Lancaster University.

[3] Grzegorczyk , M. (2013). SATFINDER. Retrieved from http://esys.com.pl/satfinder

[4] Hoots, Felix R., & Roehrich, R. L. (1980). Spacetrack Report #3: Models for Propagation of the NORAD Element Sets. Colorado Springs, CO: U.S. Air Force Aerospace Defense Command.

[5] SATELLITE AR. (2011). ANALYTICAL GRAPHICS, INC. RETRIEVED FROM HTTP://SPACEDATA.AGI.COM/MOBILEAPPS/ABOUT.HTM

[6] Vallado, D. A., Crawford, P., Hujsak, R., & Kelso, T. S. (2006). *Revisiting spacetrack report #3*. In Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference, 2006 (Vol. 3, pp. 1984–2071). (*AIAA-2006-6753)*

[7] Videotutorials YOUTUBE, stackoverflow.com

[8] WIKIPEDIA.COM

[9] http://www.developerphil.com/parcelable-vs-serializable/

[10] developer.android.com/

[11] http://stackoverflow.com/questions/5092591/what-are-the-differences-among-internal-storage-external-storage-sd-card-and-r

[12] http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/

[13] http://stackoverflow.com/questions/1995998/android-get-altitude-by-longitude-and-latitude

[14] http://gisdata.usgs.gov/xmlwebservices2/elevation_service.asmx

[15] https://developers.google.com/maps/documentation/elevation

[16] http://stackoverflow.com/questions/4699417/android-compass-orientation-on-unreliable-low-pass-filter

[17] http://stackoverflow.com/questions/12140851/sax-vs-dom-in-android

11. LIST OF ABBREVIATIONS

SDK: Software Development Kit
AR: Augmented Reality
NORAD: North American Aerospace Defence Command
TLEs: Two Line Elements Sets
XML: eXtensible Markup Language
XSD: XML Scheme Definition
SAX: Simple API for XML
SGP: Simplified General Propagation Model
SGP4: Simplified General Propagation Model Version 4
SDP4: Simplified General Deep Space Perturbation model version 4
GUI: Graphical User Interface

12. APPENDICES

APPENDIX 1: TLEs Structure

```
AAAAAAAAAAAAAAAAAAAAAAAA
1 NNNNNU NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNN +NNNNN-N +NNNNN-N N NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNN NNN.NNNN NNN.NNNN NN.NNNNNNNNNNNNN
```

Line 1	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
08	Classification (U=Unclassified)
10-11	International Designator (Last two digits of launch year)
12-14	International Designator (Launch number of the year)
15-17	International Designator (Piece of the launch)
19-20	Epoch Year (Last two digits of year)
21-32	Epoch (Day of the year and fractional portion of the day)
34-43	First Time Derivative of the Mean Motion
45-52	Second Time Derivative of Mean Motion (decimal point assumed)
54-61	BSTAR drag term (decimal point assumed)
63	Ephemeris type
65-68	Element number
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)

Line 2	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
09-16	Inclination [Degrees]
18-25	Right Ascension of the Ascending Node [Degrees]
27-33	Eccentricity (decimal point assumed)
35-42	Argument of Perigee [Degrees]
44-51	Mean Anomaly [Degrees]
53-63	Mean Motion [Revs per day]
64-68	Revolution number at epoch [Revs]
69	Checksum (Modulo 10)

```
GPS BIIA-15 (PRN 27)
1 22108U 92058A 05274.69046540 .00000036 00000-0 10000-3 0 4762
2 22108 54.7300 89.1232 0185387 243.8332 114.2955 2.00554288 95652
```

APPENDIX 2: XML Scheme


```xml
<sat>
	<info>
		<name>NOAA 1  [-]</name>
		<number>04793U</number>
	</info>
	<keplerianElements>
		<inclination>102.0640</inclination>
		<rightAsensionAscendingNode>216.2473</rightAsensionAscendingNode>
		<eccentricity>0.0032328</eccentricity>
		<argumentPerigee>114.5254</argumentPerigee>
		<meanAnomaly>245.9193</meanAnomaly>
		<meanMotion>12.5395227</meanMotion>
		<epochYear>12</epochYear>
		<epoch>189.55987732</epoch>
	</keplerianElements>
	<otherParameters>
		<meanMotionDerivate>-.00000031</meanMotionDerivate>
		<bstarDragTerm>10000E-3</bstarDragTerm>
		<ephemeridesType>0</ephemeridesType>
	</otherParameters>
</sat>
```

TECHNICAL

MANUAL

ZATDROID

Satellite Tracking and Augmented Reality App for ANDROID

1. TABLE OF CONTENTS

1. Table of Contents 3
2. List of Figures 5
3. List of Tables 7
4. Introduction 9
5. System Requirements Analysis 11
5.1. Introduction 11
5.2. Objectives 11
5.3. Requirements 12
5.3.1. Information Requirements 12
5.3.2. Functional Requirements 17
5.3.2.1. Use Case Diagram 17
5.3.2.2. Actors 17
5.3.2.3. Use Cases 18
5.3.3. Non-Functional Requirements 25
5.4. Traceability Matrix Objectives / Requirements 27
5.5. Summary 28
6. Design System Analisys 29
6.1. Introduction 29
6.2. Data Management 29
6.2.1. Conceptual Model: Entity-Relationship Model 30
6.2.2. Logical Model 31
6.2.3. Data Dictionary 31
6.3. Deployment Diagram 32
6.4. Packages Diagram 33
6.5. Classes Design 33
6.5.1. Classes Diagrams 34
6.5.2. Classes Descritpion 37
6.6. Behavior Model 53
6.6.1. Use Cases Diagram 53
6.6.2. Sequence Diagrams 54
6.7. Summary 59
7. User Interface Design 60
7.1. GUI Requirements 60
7.2. Scenarios 60
7.2.1. Scenario 1: Intro 61
7.2.2. Scenario 2a: Initial Search Menu 62
7.2.3. Scenario 2b: Internet Connection Message 63
7.2.4. Scenario 3a: Search By Key Words (Name) 64
7.2.5. Scenario 3b: Search By Type 65
7.2.6. Scenario 4: Downloading Progress Bar 66
7.2.7. Scenario 5: Satellite Names List 68
7.2.8. Scenario 6: Functionalities Main Menu 69
7.2.9. Scenario 7: GOOGLE MAPS View 70
7.2.10. Scenario 8a: Next Overhead Pass 71
7.2.11. Scenario 8b: Augmentd Reality View 72

8. Orbital Mechanics 74
8.1. Coord. Syst., newton`s & kepler`s laws. Keplerian elements. 74
8.2. NORAD Orbit Propagation Models 75
8.2.1. NORAD TLEs 75
8.2.2. NORAD Propagations Models SGP4/SPD4 76
8.3. Implementation 77
9. Breakthrough Design Features 79
9.1. Multilanguage Support 79
9.2. Multiple Screens Support: Icons, Text 79
9.3. `Asynctask` and Progress Bar: Running Code In Background. 80
9.4. Sharing and Storing Information 81
9.5. Breadcrumbs Navigation Buttons 83
9.6. GOOGLE Search Activity 83
9.7. Device Location Provider 84
9.8. Sensors Management 85
9.9. Smooth Movements Filter 86
9.10. Migration of SGP4 Code from FORTRAN / C to JAVA for ANDROID 87
9.11. Augmented Reality View: Layers Over Camera View. 87
9.12. OPENGL Usage 88
9.13. GOOGLE MAPS View 88
9.14. XML/XSD Management 89
9.15. SAX Management 90
10. Testing And Validating 91
11. List of References 92
12. List of Abbreviations 93
13. Appendices 94

2. LIST OF FIGURES

Figure 1. Use Case Diagram 17
Figure 2. Actors 17
Figure 3. Data Management Diagram 29
Figure 4. XML example 30
Figure 5. Entity – relationship model 30
Figure 6. Logical model 31
Figure 7. Deployment Diagram 32
Figure 8. Package Diagram 33
Figure 9. Colour meaning 33
Figure 10. General Classes Diagram 34
Figure 11. Satellite Explorer Classes Diagram 35
Figure 12. Augmented Reality Classes Diagram 36
Figure 13. GOOGLE MAPS Classes Diagram 37
Figure 14. Use Cases Diagram 53
Figure 15. Sequence Diagram 01: Creating sat JAVA Object from XML 54
Figure 16. Sequence Diagram 02: UC-01: Download Satellite list (search by name) 55
Figure 17. Sequence Diagram 03: UC-03: GOOGLE MAPS View 56
Figure 18. Sequence Diagram 04: UC-05: Augmented Reality View (I) 57
Figure 19. Sequence Diagram 05: UC-05: Augmented Reality View (II) 58
Figure 20. Scenarios 60
Figure 21. Scenario 1: Intro 61
Figure 22. Scenario 2: Initial search menu 62
Figure 23. Scenario 2b: Internet connection message 63
Figure 24. Scenario 3a: Search by Key Words (name) 64
Figure 25. Scenario 3b: Search by Type 65
Figure 26. Scenario 4: Downloading progress bar 66
Figure 27. Scenario 5: Satellite names list 68
Figure 28. Scenario 6: Functionalities main menu 69
Figure 28a. Scenario 6: Loading Spinner Dialog 69
Figure 29. Scenario 7: GOOGLE MAPS View 70
Figure 30. Scenario 8a: Next Overhead Pass 71
Figure 31. Scenario 8b: Augmented Reality View 72
Figure 32. Diagram for Orbit Prediction with Newton´s Equations. 74
Figure 33. TLEs Explanation 75
Figure 34. Orbital Mechanics Classes Diagram 78
Figure 35. Multilanguage Screen 79
Figure 36. BreadCrumbs 83
Figure 37. GOOGLE Search & Progress Bar 83
Figure 38. Augmented Reality View 87
Figure 39. GOOGLE MAPS View 88

3. LIST OF TABLES

Table 1. OBJ 01: Download satellites 11
Table 2. OBJ 02: Augmented reality view 11
Table 3. OBJ 03: `Google Maps` view 12
Table 4. IRQ 01: Satellite data 12
Table 5. IRQ 02: Satellite available list 13
Table 6. IRQ 03: Device orientation and georeference data 13
Table 7. IRQ 04: TLEs provider 14
Table 8. IRQ 05: Satellite types picked by the user 14
Table 9. IRQ 06: Location of XML and XSD files downloaded 15
Table 10. CRQ 01: Sat characteristics 15
Table 11. CRQ 02: Orbital parameters in TLEs 15
Table 12. CRQ 03: Sat orbital parameters calculated 15
Table 13. CRQ 04: Sat Orientation Parameters 16
Table 14. CRQ 05: Sat Georeference Parameters 16
Table 15. CRQ 06: Device parameters 16
Table 16. ACT-01: User 17
Table 17. ACT-02: Satellite TLEs provider 17
Table 18. ACT-03: Internet Service Provider 18
Table 19. ACT-04: Device sensors 18
Table 20. ACT-05: Google Maps service provider 18
Table 21. UC 01: Download Satellite TLEs 19
Table 22. UC 02: Pick a satellite 20
Table 23. UC 03: Visualize satellite in Google Maps 21
Table 24. UC 04: Retrieve device altitude with Google Maps Elevation web service 21
Table 25. UC 05: Point device at satellite with Augmented reality 23
Table 26. UC 06: Read Georeference and orientation device sensors 24
Table 27. UC 07: Validate internet connection 24
Table 28. NFR-01: Features delay limit 25
Table 29. NFR-02: Download processes must be reliable 25
Table 30. NFR-03: availability at any moment 25
Table 31. NFR-04: Compatibility with Android versions and devices 26
Table 32. NFR-05: Wifi and / or 3G (similar) 26
Table 33. NFR-06: Friendly, usable, easy and beautiful GUI 26
Table 34. Traceability Matrix Objectives / Requirements 27
Table 35. Requirements Summary 28
Table 36. Device attributes 31
Table 37. Satellite attributes 32
Table 38. intro.java 37
Table 39. listaTipoSat.java 38
Table 40. searchActivity.java 39
Table 41. searchListaSatsSAXHandler.java 39
Table 42. listaTipoSat2.java 40
Table 43. listaSats.java 41
Table 44. listaSatsSAXHandler.java 41
Table 45. satElegidoSAXHandler.java 42

Table 46. sat.java 42
Table 47. menuOpciones.java 43
Table 48. MapsActivity.java 43
Table 49. MapsOverlay.java 43
Table 50. ARIntro.java 44
Table 51. ARCameraActivityOverlay.java 44
Table 52. ARCameraPreview_Overlay.java 45
Table 53. AROurSurfaceOverlaySat.java 45
Table 54. AROurSurfaceOverlayCruz.java 45
Table 55. AROurSurfaceOverlayFlecha.java 45
Table 55a. ARGLRenderOverlayMovSat.java 46
Table 56. ARGLRenderOverlayMovCruz.java 46
Table 57. ARGLRenderOverlayMovFlecha.java 47
Table 58. ARGLSat.java 47
Table 59. ARGLCruz.java 47
Table 60. ARGLFlecha.java 48
Table 61. Vector.java 48
Table 62. DevicePositionProvider.java 49
Table 63. CalcualtionsSatPOS.java 49
Table 64. CalcualtionsSatTRACK.java 50
Table 65. CalcualtionsOrbit.java 50
Table 66. CalcualtionsOrbitSGP4.java 50
Table 67. CalcualtionsOrbitSDP4.java 51
Table 68. CalcualtionsOrbitSDP4_Deep.java 51
Table 69. CalcualtionsVarSDP4.java 51
Table 70. CalcualtionsVarGlobal.java 51
Table 71. CalcualtionsTime.java 52
Table 72. CalcualtionsMaths.java 52
Table 73. k.java 52
Table 74. visibleSats.java 52
Table 75. Design summary 59
Table 76. Classes for Orbital Mechanics Calculations 77

4. INTRODUCTION

During the years I have been studying Computer Science, it has been always on my mind the idea of joining both of my study worlds: Space Engineering and Computers.

At the moment I had to decide the purpose of my final work project, I realized that it was the time to finally develop my initial idea. After a lot of work thinking and asking friends that are working in computer science and space, I came up with this project, ZATDROID. It includes skills from both sides of my career, keeping me motived and offering me the possibility to learn topics that I have never worked.

Moreover, I found that it could be a chance for many people interested in tracking artificial satellites from their devices: tablets and smartphones. Also this App brings closer to the non-expert user the satellites we are using every day whose identity is unknown for us.

The code has been developed in JAVA for ANDROID, using Eclipse with ANDROID SDK and has been tested and designed for a *Sony Xperia Neo V* with ANDROID 2.3. It has been tested in other devices with ANDROID 4 and works perfectly.

This technical manual pretends to be a guide concerning the analysis and design of the application and a detailed description of any of the processes and calculations carried out.

5. SYSTEM REQUIREMENTS ANALYSIS

5.1. INTRODUCTION

ZATDROID offers the user the possibility of tracking any artificial satellite. It locates it in Google Maps together with its predicted trajectory and also it allows the user to see it in real time in the sky with augmented reality camera view. The application is implemented in Java for ANDROID devices (tablets or smartphones).

Orbital mechanics calculations are developed following Newton equations and NORAD (*North American Aerospace Defence Command)* propagation models, firstly published in 1980 *Spacetrack Report #3: Models for Propagation of the NORAD Element Sets* [4]

ZATDROID downloads information from a data base of satellites provided by CELESTRAK.COM, does the orbital mechanics calculations (*NORAD models*), gets the device sensors magnitudes, connects to the web service GOOGLE MAPS `Elevation` to get the altitude of the user location, processes data in terms of XML language, creates a GOOGLE MAPS views with the updated position in real time of the satellite picked and shows the position in the sky in augmented reality using OPENGL for the camera view.

The GUI (*Graphical User Interface*) manages to lead the users through an easy and friendly navigation to achieve their aims quickly, showing the results of the user picks with *BreadCrumbs*, supporting English and Spanish, showing icon-based menus and keeping the users informed of the longer processes by "*progress bars*".

5.2. OBJECTIVES

OBJ-01	**Download artificial satellites orbiting the Earth**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Description	The system will download an updated list with all the satellites that are now orbiting the Earth, with name, characteristics and orbital parameters.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	The files that contain this information are called TLEs and has a defined format. They are txt files and can be found in `www.celestrak.com` using an internet connection

Table 1. OBJ 01: Download satellites

OBJ-02	**Point the device at the updated azimuth and elevation of the satellite picked by the user with Augmented reality.**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Description	The system will calculate the elevation and azimuth of the satellite picked by solving the NORAD propagation models once azimuth, elevation, altitude, latitude and longitude of the device are obtained from the sensors. All this information will be presented into layers and over the camera view.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	Accuracy is not a big issue, 2-3 degrees error is admissible. Some visual help must be offered to lead the user to the satellite when trying to find it in the sky

Table 2. OBJ 02: Augmented reality view

OBJ-03	**Locate the picked satellite in `Google Maps`, together with its past and predicted trajectory**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Description	The system will take advantage of the GOOGLE MAPS utility to show a surprising view of the satellite onto the Earth Map updated in real time. Latitude and longitude of the satellites will be calculated using NORAD propagation models.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	Internet connection is needed.

Table 3. OBJ 03: `Google Maps` view

5.3. REQUIREMENTS

5.3.1. INFORMATION REQUIREMENTS

IRQ-01	**Information of the picked satellite with all its own data and orbital parameters**	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-01, OBJ-02, OBJ-03	
Linked requirements	UC-03, UC-05	
Description	The system will store the updated information of the satellite picked by the user with all the information retrieved from TLEs and parameters calculated within the app.	
Specific data	• Satellite characteristics • Orbital parameters in TLE • Orbital parameters calculated • Orientation Parameters • Georeference parameters	
Life time	**Average**	**Max.**
	Undefined	Undefined
Occurrences	**Average**	**Max.**
	Undefined	∞
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	The basics of this information will be first in an XML file and then all complete will be in a JAVA object "sat"	

Table 4. IRQ 01: Satellite data

IRQ-02	Information of the satellites available	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-01, OBJ-02, OBJ-03	
Linked requirements	UC-01, UC-02, UC-03, UC-05	
Description	The system will store a list with all the satellites that can be picked by the user. That is to say, all the artificial satellites orbiting the Earth at the moment.	
Specific data	List with names	
Life time	**Average**	**Max.**
	Undefined	Undefined
Occurrences	**Average**	**Max.**
	Undefined	∞
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	Only the name will be shown to the user and then all data will be retrieved.	

Table 5. IRQ 02: Satellite available list

IRQ-03	Information of the device orientation and georeference data	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-02	
Linked requirements	UC-05, UC-06	
Description	The system will obtain the updated georeference and orientation data from the sensors and eventually the altitude from a web service	
Specific data	• Longitude • Latitude • Altitude • Azimuth • Elevation	
Life time	**Average**	**Max.**
	Undefined	Undefined
Occurrences	**Average**	**Max.**
	Undefined	∞
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	Accuracy is again not a key point	

Table 6. IRQ 03: Device orientation and georeference data

IRQ-04	Information of the TLEs provider	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-01	
Linked requirements	UC-01	
Description	The system will store a list with all the links available on the internet to download TLEs as txt files. Usually it will be `www.celestrak.com`	
Specific data	Internet URL	
Life time	**Average**	**Max.**
	Undefined	Undefined
Occurrences	**Average**	**Max.**
	Undefined	∞
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	A main provider is used, but other possibilities can be searched.	

Table 7. IRQ 04: TLEs provider

IRQ-05	Information of the satellite types picked by the user	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-02, OBJ-03	
Linked requirements	UC-02	
Description	The system will store the satellite types picked by the user during the process of search. This will help the GUI to maintain a coherence format.	
Specific data	• General type • Specific type • Sat name picked	
Life time	**Average**	**Max.**
	Undefined	Undefined
Occurrences	**Average**	**Max.**
	Undefined	∞
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	`SharedPreferences` will be used to store these values.	

Table 8. IRQ 05: Satellite types picked by the user

IRQ-06	**Information of the location of the files**	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Linked objectives	OBJ-01	
Linked requirements	UC-01	
Description	The system will store XML and XSD files in the internal memory. Txt file is read, not saved.	
Specific data	File Address	
Life time	**Average**	**Max.**
	Undefined	Undefined
Occurrences	**Average**	**Max.**
	Undefined	∞
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	Only the name will be shown to the user and then all data will be retrieved.	

Table 9. IRQ 06: Location of XML and XSD files downloaded

CRQ-01	**Satellite characteristics**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	IRQ-01, UC-01
Description	The information about satellite characteristics means concepts related to the sat itself, such as name, number.
Stability	Validated
Comments	The format will be detailed in the documentation

Table 10. CRQ 01: Sat characteristics

CRQ-02	**Orbital parameters in TLEs**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	IRQ-01, UC-01
Description	The information will be retrieved from txt files downloaded
Stability	Validated
Comments	The format will be detailed in the documentation

Table 11. CRQ 02: Orbital parameters in TLEs

CRQ-03	**Sat orbital parameters calculated**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	IRQ-01, UC-03, UC-05
Description	The information calculated with NORAD propagation models about orbit parameters: position, velocity. They are vectors and in International System of Units.
Stability	Validated
Comments	The format will be detailed in the documentation

Table 12. CRQ 03: Sat orbital parameters calculated

CRQ-04	Sat orientation parameters
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	IRQ-01, UC-03, UC-05
Description	The information about satellite orientation parameters consists of azimuth (-180º to 180º), elevation (-90º to 90º)
Stability	Validated
Comments	The format will be detailed in the documentation

Table 13. CRQ 04: Sat Orientation Parameters

CRQ-05	Sat Georeference parameters
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	IRQ-01, UC-03, UC-05
Description	The information about georeference consist of latitude (-90º to 90º), longitude (-180º to 180º) and altitude (km)
Stability	Validated
Comments	The format will be detailed in the documentation

Table 14. CRQ 05: Sat Georeference Parameters

CRQ-06	Device Latitude, longitude, altitude, azimuth and elevation
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-02
Linked requirements	IRQ-03, UC-05
Description	The information concerning georeference and orientation consists of these 5 parameters. Degrees with restrictions as usual.
Stability	Validated
Comments	The format will be detailed in the documentation

Table 15. CRQ 06: Device parameters

5.3.2. FUNCTIONAL REQUIREMENTS

5.3.2.1. USE CASE DIAGRAM

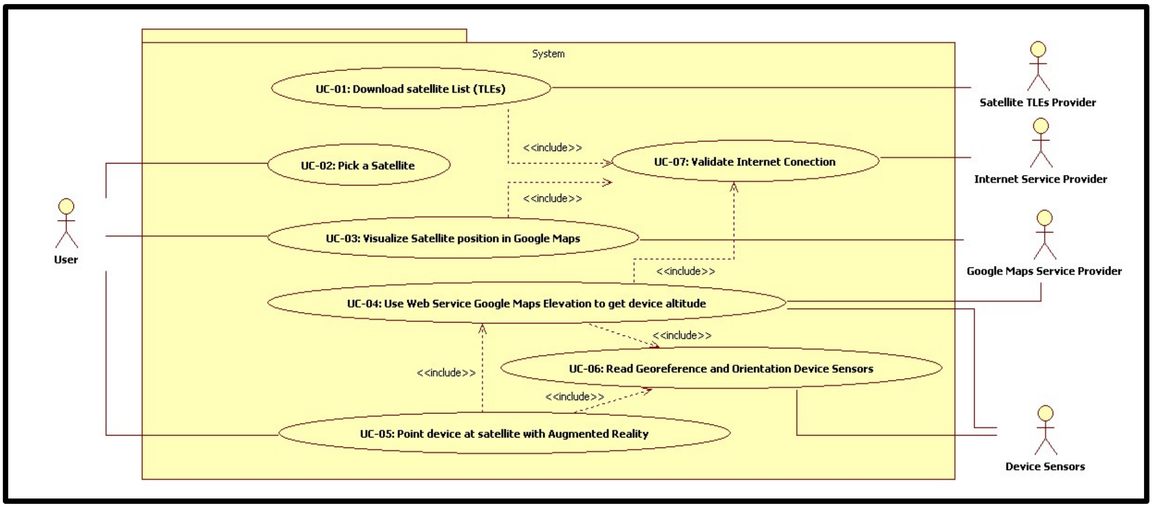


Figure 1. Use Case Diagram

5.3.2.2. ACTORS

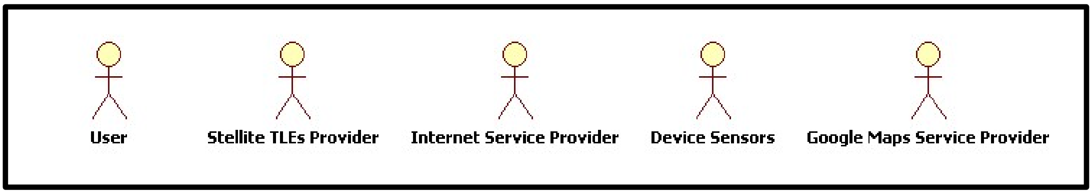


Figure 2. Actors

ACT–01	User
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Description	Represents the user that runs the application. No training or studies are required to understand ZATDROID.

Table 16. ACT-01: User

ACT–02	Satellite TLEs provider
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Description	It is usually `www.celestrak.com.` A web site that is maintained by T.S. Kelso and that offers freely TLEs everyday updated.

Table 17. ACT-02: Satellite TLEs provider

ACT–03	Internet Service provider
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Description	Any provider is allowed, wifi, network…

Table 18. ACT-03: Internet Service Provider

ACT–04	Device sensors
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Description	Accelerometer, gravity, magnetic, GPS and network are used by ZATDROID

Table 19. ACT-04: Device sensors

ACT–05	GOOGLE MAPS service provider
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Description	It provides GOOGLE MAPS view functionality and web service to get altitude in case needed

Table 20. ACT-05: Google Maps service provider

5.3.2.3. USE CASES

UC-01	Download Satellite list (TLEs)	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-01	
Linked requirements	IRQ-01, IRQ-02, IRQ-04, IRQ-05, IRQ-06, UC-02	
Description	The system will read txt files and create XML according to XSD. The information of TLEs will be described.	
Precondition	• Internet connection available by ACT-03 • Search by key word performed by ACT-01 • Specific type picked by ACT-01	
Normal Sequence	**Step**	**Action**
	1a	If search by type is performed, the Specific type picked is used to select the URL from ACT-02 to download *txt* files.
	1b	If search by key words is performed, all *txt* files matched with a list to get URL from ACT-02 are to be downloaded
	2	An `Asynctask` is started (it allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers)
	3	It is checked that ACT-02 is working
	4	A progress bar is shown to the user to indicate that a download is being carried out. The bar is updated with percentage of the total achieved.
	5	In the background, the download process is performed. A URL object is created

	6	With help of `BufferReader` and `InputStreamReader` the *txt* file is read.
	7	While *txt* is being read, XML is being created (*based on its XSD*)
	8	Progress bar is filled 100%. Conflictive download process finished
	9	Once XML file is finished, a search is performed with SAX to retrieve only the names that are to be shown to the user.
	10	`DefaultHandler` is inherited using its classes `elementRoot` and `Element` to search through the XML quickly, without creating a complete and large DOM, for the satellite names
Post-condition	A `ListView` with the names of the satellites from the satellite type picked by the user or the sat names that fit key words typed by the user	
Exception	**Step**	**Action**
	1	If ACT-02 is not working, the process is cancelled and a message is shown
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	`Celestrak.com` has the txt files in URL such as `www.celestrak.com/…/weather.txt.` These are URL which contains *txt* files with TLEs classified by satellite type. So it is needed to know the type to be able to download the file.	

Table 21. UC 01: Download Satellite TLEs

UC-02	**Pick a Satellite**	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Sources	-	
Linked objectives	OBJ-01	
Linked requirements	IRQ-01, IRQ-02, IRQ-04, IRQ-05, UC-01	
Description	The user is required to pick a satellite given its name. The search can be performed in two ways: by key words or by type.	
Precondition	• Internet connection available by ACT-03	
Normal Sequence	**Step**	**Action**
	1	There are two ways to pick a satellite: search by key words (a) or search by satellite type (b)
	2a	A GOOGLE Search is started with the default menu for ANDROID
	3a	Key words are typed by the user to search the name of the sat
	4a	The download process is performed as described in UC-01
	5a	A `ListView` with all satellites whose name fits the key words is shown.
	2b	The user picks one of the 6 icons with text (*English or Spanish*) that indicates general satellite types
	3b	The General Type is stored in `SharedPreferences` so that next activities can read it
	4b	Another screen is shown customized according the general type previously picked, retrieved from `SharedPreferences`
	5b	The user picks a specific sat type and it is stored again in `SharedPreferences.`
	6b	The download process is performed as described in UC-01
	7b	A `ListView` with all satellites from the type picked is shown.

	end1	The User picks any of them
	end2	A SAX method searches the XML for the sat picked and retrieves the TLE information from this satellite. A “sat” JAVA object is created as `Parcelable`, so that other activities can access it.
Post-condition	A satellite picked by the user from a list with satellites names and a “sat” JAVA object created with information of the satellite picked	
Exception	**Step**	**Action**
	1	If `celestrak.com` is not working, the process is cancelled and a message is shown
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	-	

Table 22. UC 02: Pick a satellite

UC-03	**Visualize Satellite position in GOOGLE MAPS**	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Linked objectives	OBJ-03	
Linked requirements	IRQ-01, UC-02	
Description	Taking advantage of GOOGLE MAPS View ZATDROID positions an icon representing the satellite picked in real time, together with its trajectory.	
Precondition	• Internet connection available by ACT-03 • Sat JAVA object created with information of the satellite picked	
Normal Sequence	**Step**	**Action**
	1	`MapsActivity`, which extends `com.google.android.maps.mapView` provides by ACT-05, starts the maps layout and retrieves “sat” object
	2	Compass is added as a layer to the `MapView`
	3	Also a `MapOverlay` class is created as a layer
	4	Current time is obtained and modified by `CalculationsTime`
	5	`CalculationsSatPOS` takes current time and *sat* object and calculate latitude, longitude and altitude of satellite, implementing NORAD propagation models SGP4 / SDP4. Firstly, current position and velocity, and then subsatellite point with latitude and longitude. Many classes are used, names starts with `Calculations[…].` More detailed description of this step will be explained in section 8.
	6	A `GeoPoint` is created with latitude and longitude and an icon is represented and added as a layer to the `MapView`
	7	There is a loop calculating latitude and longitude of satellite from 10 minutes in the past and 10 minutes predicted for the future. A line is drawn with these points and added as a layer to the `MapView`
	8	A `TextView` is also added as a layer, with name, latitude, longitude and altitude of the satellite.
	9	The `onDraw` method is refreshed instantly and redraws the complete `MapView`, with all the layers: icon, trajectory and text. Trajectory is delayed to be refreshed only every second. All the calculations are too complicated to be refreshed instantly. The tests performed indicate the line is deformed in some seconds.

<table>
<tr><td>Post-condition</td><td colspan="2">An icon representing the satellite is shown in GOOGLE MAPS, updated instantly together with its past (10 min) and (10 min) predicted trajectory and text with name, latitude, longitude and altitude are shown to create a real time view.</td></tr>
<tr><td rowspan="3">Exception</td><td>Step</td><td>Action</td></tr>
<tr><td>1</td><td>If the calculation takes more time than expected, it stops and shows a message</td></tr>
<tr><td>2</td><td>The trajectory is updated every second. This delay is due to tests carried out indicating that calculations cannot be developed quickly. The line would otherwise be deformed</td></tr>
<tr><td>Importance</td><td colspan="2">High</td></tr>
<tr><td>Urgency</td><td colspan="2">High</td></tr>
<tr><td>State</td><td colspan="2">Validated</td></tr>
<tr><td>Stability</td><td colspan="2">High</td></tr>
</table>

Table 23. UC 03: Visualize satellite in GOOGLE MAPS

<table>
<tr><td>UC-04</td><td colspan="2">Use web service GOOGLE MAPS elevation to get device altitude</td></tr>
<tr><td>Version</td><td colspan="2">V1.0 – 15/08/2013</td></tr>
<tr><td>Authors</td><td colspan="2">Rodrigo Santos Álvarez</td></tr>
<tr><td>Linked objectives</td><td colspan="2">OBJ-02</td></tr>
<tr><td>Linked requirements</td><td colspan="2">IRQ-03, UC-06</td></tr>
<tr><td>Description</td><td colspan="2">If GPS is not working, a web service is called to retrieve device altitude. It is needed to calculate azimuth and elevation of the satellite.</td></tr>
<tr><td>Precondition</td><td colspan="2">• Internet connection available by ACT-03
• Device Sensors provide latitude and longitude of sat by ACT-04
• GPS not available. Network available</td></tr>
<tr><td rowspan="4">Normal Sequence</td><td>Step</td><td>Action</td></tr>
<tr><td>1</td><td>If GPS is not available, then altitude is retrieved from web Service. Nevertheless, network must be available. Latitude and longitude are needed to send them to the web service.</td></tr>
<tr><td>2</td><td><code>HttpClient, HttpContext</code> are called to establish the connection. URL is created with latitude and longitude from device <code>http://maps.googleapis.com/maps/api/elevation/xml?locations= “latitude”,”longitude”&sensor=true</code></td></tr>
<tr><td>3</td><td><code>HttpGet</code> sends this URL to the web service. <code>HttpResponse</code> and <code>HttpEntity</code> process the answer. It is a XML file and the altitude is written between <<i>elevation</i>> tags</td></tr>
<tr><td>Post-condition</td><td colspan="2">Device altitude is retrieved and used to calculate azimuth and elevation of the satellite</td></tr>
<tr><td rowspan="2">Exception</td><td>Step</td><td>Action</td></tr>
<tr><td>1</td><td>If web service is not working, a message is shown</td></tr>
<tr><td>Importance</td><td colspan="2">High</td></tr>
<tr><td>Urgency</td><td colspan="2">High</td></tr>
<tr><td>State</td><td colspan="2">Validated</td></tr>
<tr><td>Stability</td><td colspan="2">High</td></tr>
<tr><td>Comments</td><td colspan="2">There are two web services that can provide altitude out of latitude and longitude:
• USGS Elevation Query Web Service:
<code>http://gisdata.usgs.gov/xmlwebservices2/elevation_service.asmx</code>
• GOOGLE MAPS Elevation API Web Service:
<code>https://developers.google.com/maps/documentation/elevation/</code>
GOOGLE MAPS elevation is used in ZATDROID</td></tr>
</table>

Table 24. UC 04: Retrieve device altitude with Google Maps Elevation web service

<table>
<tr><th>UC-05</th><th colspan="2">Point Device at satellite with Augmented Reality</th></tr>
<tr><td>Version</td><td colspan="2">V1.0 – 15/08/2013</td></tr>
<tr><td>Authors</td><td colspan="2">Rodrigo Santos Álvarez</td></tr>
<tr><td>Sources</td><td colspan="2">-</td></tr>
<tr><td>Linked objectives</td><td colspan="2">OBJ-02</td></tr>
<tr><td>Linked requirements</td><td colspan="2">IRQ-01, IRQ-03, IRQ-05, UC-06, UC-02</td></tr>
<tr><td>Description</td><td colspan="2">The device camera will show an icon in the right position in the sky (azimuth and elevation) representing the sat picked by the user. Layers over the camera view show information and allows the user to rotate the device to search the sat. It is indeed the largest of the use cases.</td></tr>
<tr><td>Precondition</td><td colspan="2">• Device Sensors provide latitude, longitude, altitude, azimuth and elevation of device by ACT-04
• Sat Java object created with information of the satellite picked</td></tr>
<tr><td rowspan="15">Normal Sequence</td><td>Step</td><td>Action</td></tr>
<tr><td>1</td><td><code>ARIntro</code> runs <code>DevicePositionProvider</code> to get latitude, longitude and altitude of device. (UC-06)</td></tr>
<tr><td>2</td><td><code>LocationManager</code> is used to retrieve latitude and longitude from GPS or network.</td></tr>
<tr><td>3</td><td>If GPS is available, altitude is known easily. If not, Web service is called as described in UC-04.</td></tr>
<tr><td>4</td><td><code>ARIntro</code> continues running <code>CalculationsSatTRACK to</code> calculate Azimuth and elevation of satellite.</td></tr>
<tr><td>5</td><td><code>CalculationsSatTRACK</code> takes current time, sat object and device georeference ad orientation and implements NORAD propagation models SGP4 / SDP4. Firstly, current position and velocity, and then position in the sky. Many classes are used, names starts with <code>Calculations[…].</code> More detailed description of this step will be explained in section 8.</td></tr>
<tr><td>6</td><td>If sat elevation is less than 0, satellite picked is under the horizon. A loop runs to calculate next overhead pass. Then an <code>alertDialog</code> pops up with such information.</td></tr>
<tr><td>7</td><td>ARCameraActivityOverlay is executed</td></tr>
<tr><td>8</td><td>Here <code>SensorManager</code> and <code>Sensor</code> Classes retrieve all data from accelerometer, magnetic and gravity sensors to calculate the orientation (elevation and azimuth) of the device. <code>RotationMatrix</code> and <code>OrientationMatrix</code> are used.</td></tr>
<tr><td>9</td><td>Elevation and azimuth of the device are passed to the Sat, Cruz and Flecha <code>Renders</code>. Gravity vector is passed to sat <code>Render</code>.</td></tr>
<tr><td rowspan="3">10</td><td>Two TextViews are layers overlaying the camera view:</td></tr>
<tr><td>One is showing azimuth, elevation and name of the satellite</td></tr>
<tr><td>The other is showing elevation and azimuth of the device in real time, as the user rotates the device the <code>textView</code> is updated.</td></tr>
<tr><td>11</td><td>3 Layers that will overlay <code>CameraView</code> are created. Each layer consists of <code>GLSurfaceView</code>, <code>Render</code> and object class and it is set transparent.</td></tr>
<tr><td>12</td><td><code>CameraView: (SurfaceView)</code> <code>ARCameraPreview_Overlay</code>. Initiates camera, where augmented reality makes sense.</td></tr>
</table>

	13	GLSurfaceView: AROurSurfaceOverlaySat icon of the satellite.
		Render: ARGLRenderOverlayMovSat Using GL10 object (OPENGL), it customizes the virtual camera showing the icon (GLU.gluLookAt), refreshed the view with data coming from sensors, smooth the refreshing with a filter and calculates a transformation matrix from device coordinate system into Earth Coordinate system.
		JAVA Object: ARGLSat Creates textures, bitmaps, and geometry of satellite icon
	14	GLSurfaceView: AROurSurfaceOverlayCruz. Central square.
		Render: ARGLRenderOverlayMovCruz Similar to previous render, using GL10 visualize the central square and it changes the colour and size when sat is just in the centre of the camera view.
		JAVA Object: ARGLCruz Creates textures, bitmaps, and geometry of the square.
	15	GLSurfaceView: AROurSurfaceOverlayFlecha: Arrow indicating the direction of the satellite in the sky.
		Render: ARGLRenderOverlayMovFlecha Again, GL10 is used to update in real time the direction of the arrow, pointing always at the satellite. It helps the user to find the sat, as it tell information of how to rotate the device.
		JAVA Object: ARGLFlecha Creates textures, bitmaps, and geometry of the arrow
Post-condition	The icon, device information, sat information, arrow and textViews with information are shown over the camera view. The icon can be found only if the device is pointing at the elevation and azimuth of the sat.	
Exception	**Step**	**Action**
	1	The OPENGL does not support sometimes so many things and updates. Then the activity is reset.
Importance	High	
Urgency	High	
State	Validated	
Stability	High	
Comments	The accuracy is not high.	

Table 25. UC 05: Point device at satellite with Augmented reality

UC-06	**Read Georeference and orientation device sensors**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-02
Linked requirements	IRQ-03, UC-04
Description	Georeference (latitude, longitude, altitude) is implemented in class DevicePositionProvider and used to calculate elevation and azimuth of the sat. The sat orientation in the sky depends on the position of the device in the surface of the Earth. GPS and/or network are required. Orientation (elevation and azimuth) is implemented inside the ARCameraActivityOverlay and it is used to match device and sat orientation. Accelerometer, magnetic and gravity sensors are required
Precondition	Device with sensors accelerometer, Magnetic, Gravity, GPS and network (ACT-04)

Normal Sequence		**Step**	**Action**
	Georeference	1	`DevicePositionProvider` class is called
		2	With `LocationManager` class, GPS and network are started
		3	Firstly, latitude and longitude are retrieved from network provider. GPS is slower and consumes more battery.
		4	Secondly, if GPS is available and active, latitude and longitude are again retrieved and refreshed, as GPS is usually more precise.
		5	If GPS is active and available, altitude is easily found
		6	If not, and network is active, UC-04 retrieves the altitude from a web service.
		7	Altitude: [km], Latitude: [90º, -90º], Longitude: [-180º, 180º]
	Orientation	1	`ARCameraActivityOverlay` class is called
		2	With `SensorManager` and Sensor classes, magnetic, accelerometer and gravity sensors are started
		3	The `onSensorChanged` class notifies every change in the magnitude offered by each sensor.
		4	`RotationMatrix` and `OrientationMatrix` are obtained to calculate elevation and azimuth in degrees with mathematical transformations
		5	Elevation: [-90º, 90º], Azimuth: [-180º, 180º]
Post-condition	Georeference (latitude, longitude, altitude) and orientation (azimuth and elevation) of the device are available.		
Exception	**Step**	**Action**	
	1	If GPS is not available, a web service is connected (UC-04)	
	2	If GPS is not active, a message pops up to require its activation	
Importance	High		
Urgency	High		
State	Validated		
Stability	High		
Comments	The accuracy is not high.		

Table 26. UC 06: Read Georeference and orientation device sensors

UC-07	**Validate internet connection**	
Version	V1.0 – 15/08/2013	
Authors	Rodrigo Santos Álvarez	
Linked objectives	OBJ-01, OBJ-03	
Linked requirements	IRQ-02, IRQ-04, UC-01, UC-03	
Description	Internet must be available so that all functions can work. Wifi or 3G or similar: ACT-03	
Precondition	-	
Normal Sequence	**Step**	**Action**
	1	Starts `ConnectivityManager`
	2	Checks that at least one internet connection is active
Post-condition	Internet is available and active	
Exception	**Step**	**Action**
	1	If there is no internet connection, a message is shown and the app does not starts.
Importance	High	
Urgency	High	
State	Validated	
Stability	High	

Table 27. UC 07: Validate internet connection

5.3.3. NON-FUNCTIONAL REQUIREMENTS

Performance:

NFR–01	**Features delay must be less than 3 seconds**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	IRQ-01, IRQ-02, IRQ-03, IRQ-04, IRQ-05, UC-01, UC-03, UC-05
Description	The application must be quick. Delays are not desirable when navigating. Downloads are the longest processes. Tests must be carried out with different versions and devices.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	Only one could be longer than 3 seconds (downloading all TLEs when performing a search by key words)

Table 28. NFR-01: Features delay limit

Reliability:

NFR–02	**Download processes must be reliable**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01
Linked requirements	IRQ-02, IRQ-04, UC-01
Description	`ASynctask` must be used to assure the reliability of these processes
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	-

Table 29. NFR-02: Download processes must be reliable

Availability:

NFR–03	**Availability at any moment**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	-
Description	ACT-02, ACT-03 and ACT-05 availability makes this app dependant. Usually there will be no problem at all and it will be used at any time, but if any of these actors does not give support, application cannot run.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	-

Table 30. NFR-03: availability at any moment

Compatiblity:

NFR–04	**Compatibility with Android versions and devices**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	-
Description	The code must run on all devices types with ANDROID 2.3 and forward.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	-

Table 31. NFR-04: Compatibility with Android versions and devices

Compatibility:

NFR–05	**Wifi and / or 3G (similar)**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	-
Description	Wifi and/or 3G will be a must in order to work with this app.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	-

Table 32. NFR-05: Wifi and / or 3G (similar)

GUI *(Graphical User Interface)*:

NFR–06	**Friendly, usable, easy and beautiful GUI**
Version	V1.0 – 15/08/2013
Authors	Rodrigo Santos Álvarez
Sources	-
Linked objectives	OBJ-01, OBJ-02, OBJ-03
Linked requirements	UC-01, UC-03, UC-05
Description	As it is the feature facing the user, GUI must fit all the requirements to make the app attractive.
Importance	High
Urgency	High
State	Validated
Stability	High
Comments	-

Table 33. NFR-06: Friendly, usable, easy and beautiful GUI

5.4. TRACEABILITY MATRIX OBJECTIVES / REQUIREMENTS

	OBJ-01	OBJ-02	OBJ-03
IRQ-01	X	X	X
IRQ-02	X	X	X
IRQ-03		X	
IRQ-04	X		
IRQ-05		X	X
IRQ-06	X		
CRQ-01	X	X	X
CRQ-02	X	X	X
CRQ-03	X	X	X
CRQ-04	X	X	X
CRQ-05	X	X	X
CRQ-06			X
UC-01	X		
UC-02	X		
UC-03			X
UC-04		X	
UC-05		X	
UC-06		X	
UC-07	X		X
NFR-01	X	X	X
NFR-02	X		
NFR-03	X	X	X
NFR-04			
NFR-05	X	X	X
NFR-06	X	X	X

Table 34. Traceability Matrix Objectives / Requirements

5.5. SUMMARY

TYPE	ID	DESCRIPTION
ACTORS	ACT-01	User
	ACT-02	Satellite TLEs provider
	ACT-03	Internet Service provider
	ACT-04	Device sensors
	ACT-05	GOOGLE MAPS service provider
OBJECTIVES	OBJ-01	Download artificial satellites orbiting the Earth
	OBJ-02	Point the device at the updated azimuth and elevation of the satellite picked by the user with Augmented reality.
	OBJ-03	Locate the picked satellite in `Google Maps`, together with its past and predicted trajectory
INFORMATION REQUIREMENTS	IRQ-01	Information of the picked satellite with all its own data and orbital parameters
	IRQ-02	Information of the satellites available
	IRQ-03	Information of the device orientation and georeference data
	IRQ-04	Information of the TLEs provider
	IRQ-05	Information of the satellite types picked by the user
	IRQ-06	Information of the location of the files
	CRQ-01	Satellite characteristics
	CRQ-02	Orbital parameters in TLEs
	CRQ-03	Sat orbital parameters calculated
	CRQ-04	Sat orientation parameters
	CRQ-05	Sat Georeference parameters
	CRQ-06	Device Latitude, longitude, altitude, azimuth and elevation
FUNCTIONAL REQUIREMENTS – USE CASES	UC-01	Download Satellite list (TLEs)
	UC-02	Pick a Satellite
	UC-03	Visualize Satellite position in GOOGLE MAPS
	UC-04	Use web service GOOGLE MAPS elevation to get device altitude
	UC-05	Point Device at satellite with Augmented Reality
	UC-06	Read Georeference and orientation device sensors
	UC-07	Validate internet connection
NON-FUNCTIONAL REQUIREMENTS	NFR-01	Features delay must be less than 3 seconds
	NFR-02	Download processes must be reliable
	NFR-03	Availability at any moment
	NFR-04	Compatibility with Android versions and devices
	NFR-05	Wifi and / or 3G (similar)
	NFR-06	Friendly, usable, easy and beautiful GUI

Table 35. Requirements Summary

6. DESIGN SYSTEM ANALISYS

6.1. INTRODUCTION

After analysing the system, now it is time to design all the procedures to finally get all that functionalities. The Android architecture is based on MVC framework (Model – View - Controller). So layouts, classes and activities help the programmer to create code easily.

6.2. DATA MANAGEMENT

The data management in this project uses no Relational Data Base. It is not considered necessary to create such complex concept because there are not enough entities and relationships. Android implements SQLite to make the process of creation and management a Data Base easy. Instead, the two entities has been studied from another point of view Also `SharedPreferences` option to share data within the code is mentioned.

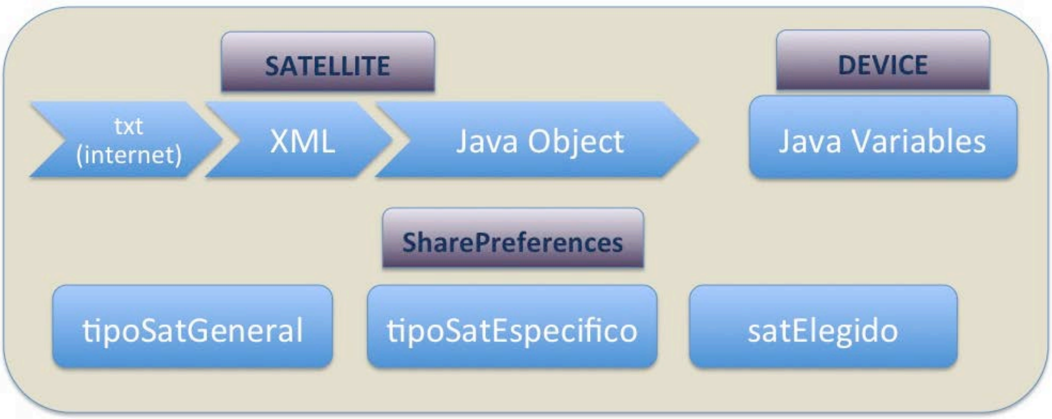


Figure 3. Data Management Diagram

As Figure 3 shows:

Satellite:

- Parameters and data from TLEs begin in txt format, downloaded from the web, via www.celestrak.com. These files are classified in terms of satellite types, so a lot of satellites are inside the file.
- Then the system parses the txt and creates an XML (with its XSD associated). The structure of this XML file can be seen in Figure 4. XML has become widely common when management data on the internet and all programs. Therefore, one of the objectives at the beginning of this project was to learn XML. Moreover, Android implements SAX as a way to manage data and makes it really easy.
- Finally, to satisfy the need of the application, only one picked satellite will be needed. With help of SAX search, the TLE information from the picked satellite is extracted and a Java Object is created with TLEs parameters and more. Future calculations concerning satellite position, velocity, longitude, latitude, altitude, elevation, azimuth… will be stored in this object.

Device:

- Latitude, longitude, altitude, azimuth and elevation are the parameters that have to be taken into account. No JAVA object or XML is created. These values are treated individually and tracked during the methods easily.
- Another solution would have been to create a JAVA Object. It has not been considered because it would have added complexity to the code and according to the author´s view, it is not necessary.

```
<sat>
	<info>
		<name>NOAA 1 [-]</name>
		<number>04793U</number>
	</info>
	<keplerianElements>
		<inclination>102.0640</inclination>
		<rightAsensionAscendingNode>216.2473</rightAsensionAscendingNode>
		<eccentricity>0.0032328</eccentricity>
		<argumentPerigee>114.5254</argumentPerigee>
		<meanAnomaly>245.9193</meanAnomaly>
		<meanMotion>12.5395227</meanMotion>
		<epochYear>12</epochYear>
		<epoch>189.55987732</epoch>
	</keplerianElements>
	<otherParameters>
		<meanMotionDerivate>-.00000031</meanMotionDerivate>
		<bstarDragTerm>10000E-3</bstarDragTerm>
		<ephemeridesType>0</ephemeridesType>
	</otherParameters>
</sat>
```

Figure 4. XML example

SharedPreferences:

In the process of picking one satellite, the user is prompted several screens. Every screen implements activity and the options selected must be stored to create *BredCrumbs* and navigate through the classification. There are three strings stored:

- `tipoSatGeneral`: First selection from 6 general types.
- `tipoSatEspecífico`: second selection, with final specific type.
- `satElegido`: name of the satellite picked.

6.2.1. CONCEPTUAL MODEL: ENTITY-RELATIONSHIP MODEL

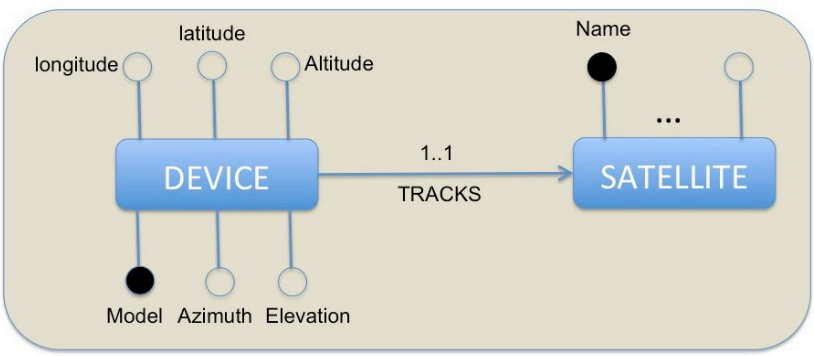


Figure 5. Entity – relationship model

Although there is no relational Data Base, this relationship between device and satellite can be set.

- One entity device is tracking only one entity satellite, after picking one from all the satellites listed in the complete XML file.
- The device is unique as there is only one device executing the application. Anyway, the model could be the primary key, just as a help.
- All satellite attributes are explained in table 37 in data dictionary section.

6.2.2. LOGICAL MODEL

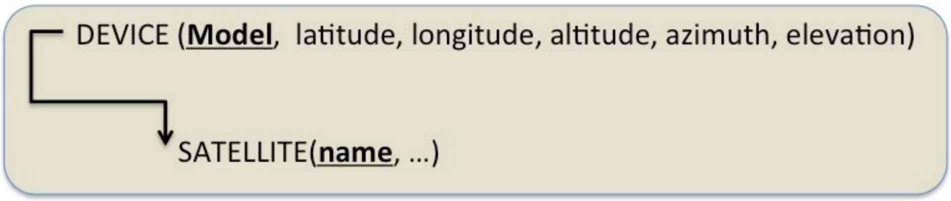


Figure 6. Logical model

ZATDROID will have an only one device and one satellite every time the user picks one satellite, but satellites will not be saved and "sat" Java object will change its values with each new satellite picked. There will be no records as in a relational database.

6.2.3. DATA DICTIONARY

ENTITY: DEVICE		
Attribute	**Origin**	**Description**
lat	**devidePositionProvider**: GPS, network, web Service Google Maps Elevation	latitude [-90º, 90º]
lon		longitude [-180º, 180º]
alt		altitude [km]
az	**Sensors** accelerometers, magnetic, gravity	azimuth [-180º, 180º]
ele		elevation [-90º, 90º]

Table 36. Device attributes

ENTITY: SATELLITE		
Attribute	**type**	**Description**
r1	calculated with NORAD propagation model SGP4 / SPD4	Position x (km)
r2		position y (km)
r3		position zv (km)
rdot1		velocity x (m/s)
rdot2		velocity y (m/s)
rdot3		velocity z (m/s)
lat		latitude [-90º, 90º]
lon		longitude [-180º, 180º]
az		azimuth [-180º, 180º]
alt		altitude [km]
range		distance *(sat & device)* [km]
ele		elevation [-90º, 90º]

name	TLEs data imported	name (unique)
number		-
inclination		inclination orbit plane [-90º, 90º]
RAAN		Right ascension of ascending node
eccentricity		eccentricity of orbit [0, ∞]
argumentPerigee		argument Perigee orbit [-180º, 180]
meanAnomaly		-
meanMotion		-
meanMotionDerivate		-
bstarDragsTerm		drag term
ephemerides type		NORAD TLE type [0-8]
encontrado	aux	[true, false]
epoch	Time Calculations: moment in time when TLEs parameters are measured	Day and minutes (NORAD format)
epochYear		year (NORAD format)
epochC		date in usual format
epochN		date (NORAD format)
epochJD		date (Julian Date)

Table 37. Satellite attributes

6.3. DEPLOYMENT DIAGRAM

Figure 7 shows the interaction between software and hardware elements.

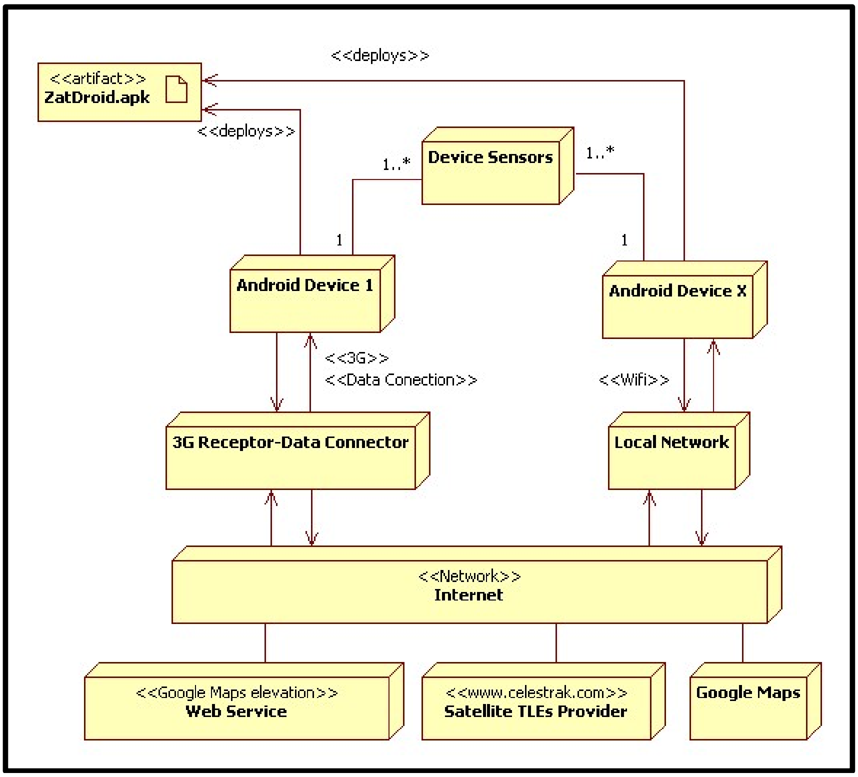


Figure 7. Deployment Diagram

6.4. PACKAGES DIAGRAM

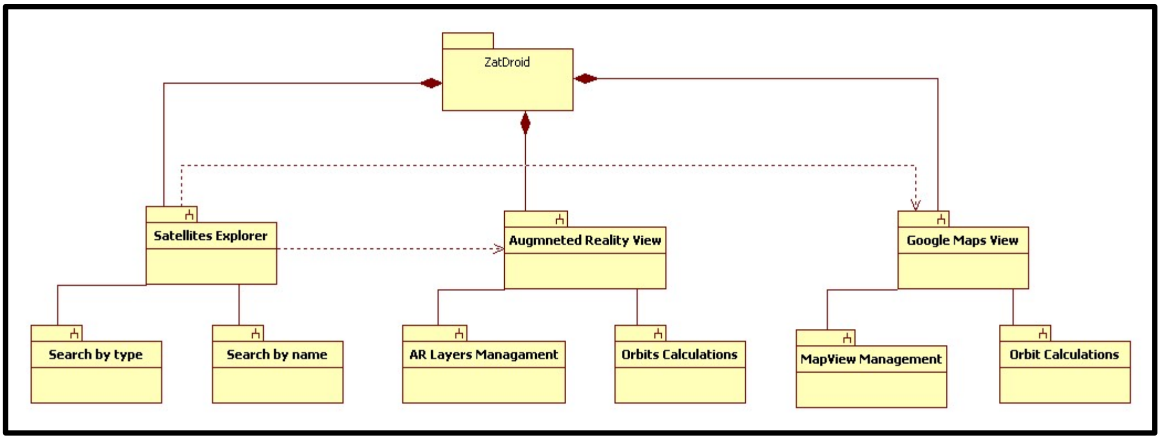


Figure 8. Package Diagram

- **Satellite Explorer:**
  It is the functionality where the TLEs are downloaded and the satellite is picked by the user, with two options: by type, by name (key words)

- **Augmented reality View:**
  Position satellite in the sky using augmented reality

- **Google Maps View**:
  Location satellite in a `mapView`

6.5. CLASSES DESIGN

Next step is unpacking and describing classes and their relationships. A general diagram firstly introduces the application scheme. Then every package is described as a subsystem.
Android classes has been included in the diagrams as a help to follow the routes and functionalities. Needless to say that not all android classes, only the most relevant ones. Android classes have blue colour while own classes are yellow.
It is also necessary to mention that attributes and methods are not added to the diagrams in order to clarify the understanding. They are described afterwards in the tables, though.

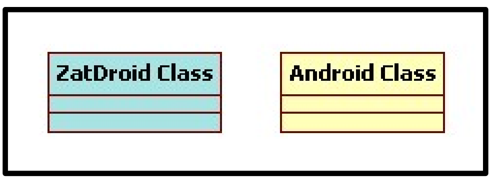


Figure 9. Colour meaning

6.5.1. CLASSES DIAGRAMS

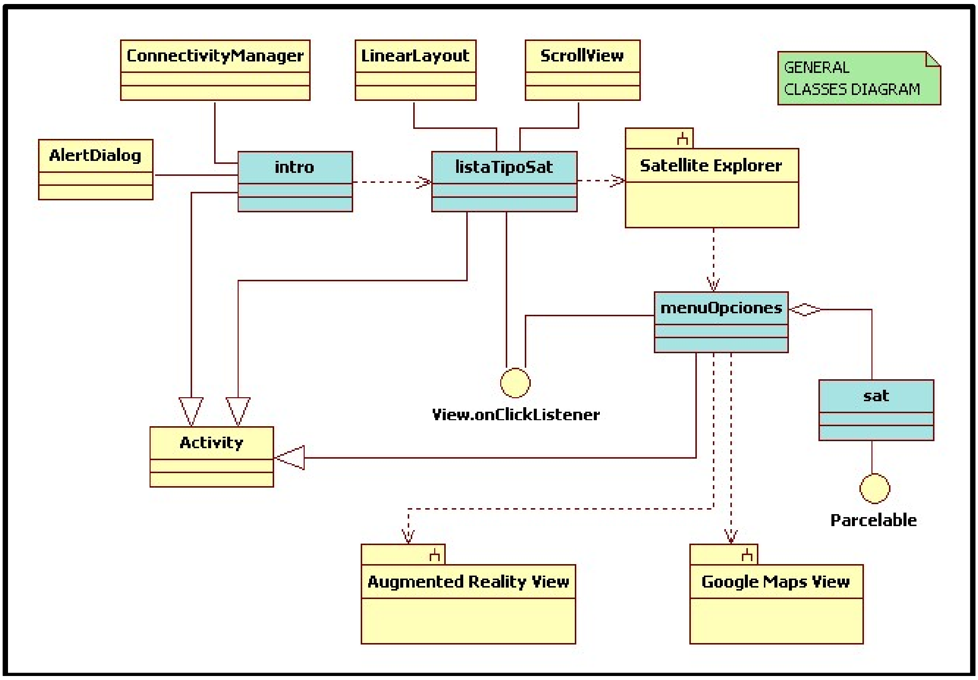


Figure 10. General Classes Diagram

Figure 10 describes an overview of the complete system. It gives information about the starting point and the options that the user can see. Subsystems will be detailed next.

Once the application is started, the user must decide whether searching a satellite by its type or by its name (key words), figure 11. Both ways implement:

- SAX handing (XML Android search API) and XML file creation.
- `MiTarea` as an `Asyntask` to manage the download of the TLEs in background and show a `progressDialog` to inform the user of the process
- `Parcelable` "sat" object is created so that this information in stored for future functions.
- When the user finally picks a satellite, the functionalities menu is opened; with AR view or GOOGLE MAPS view need to be selected.

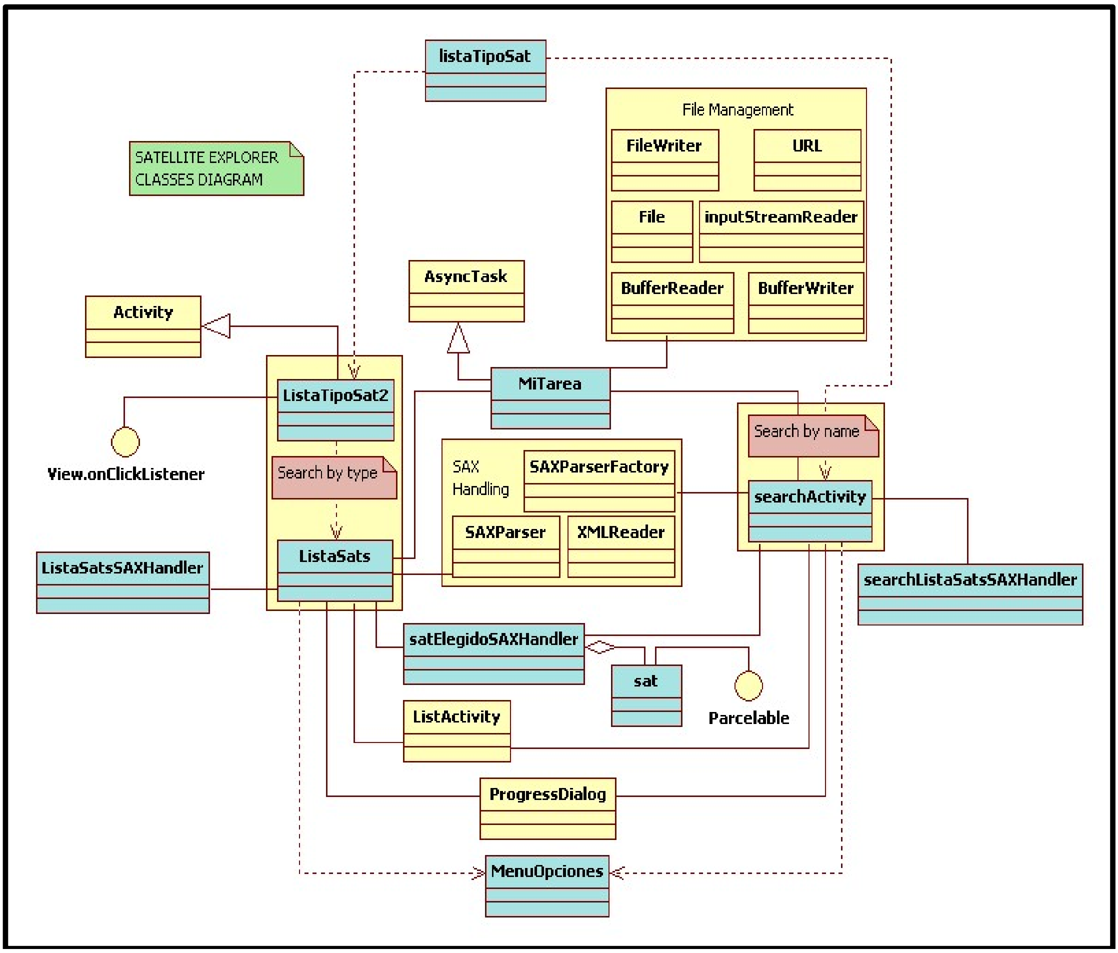


Figure 11. Satellite Explorer Classes Diagram

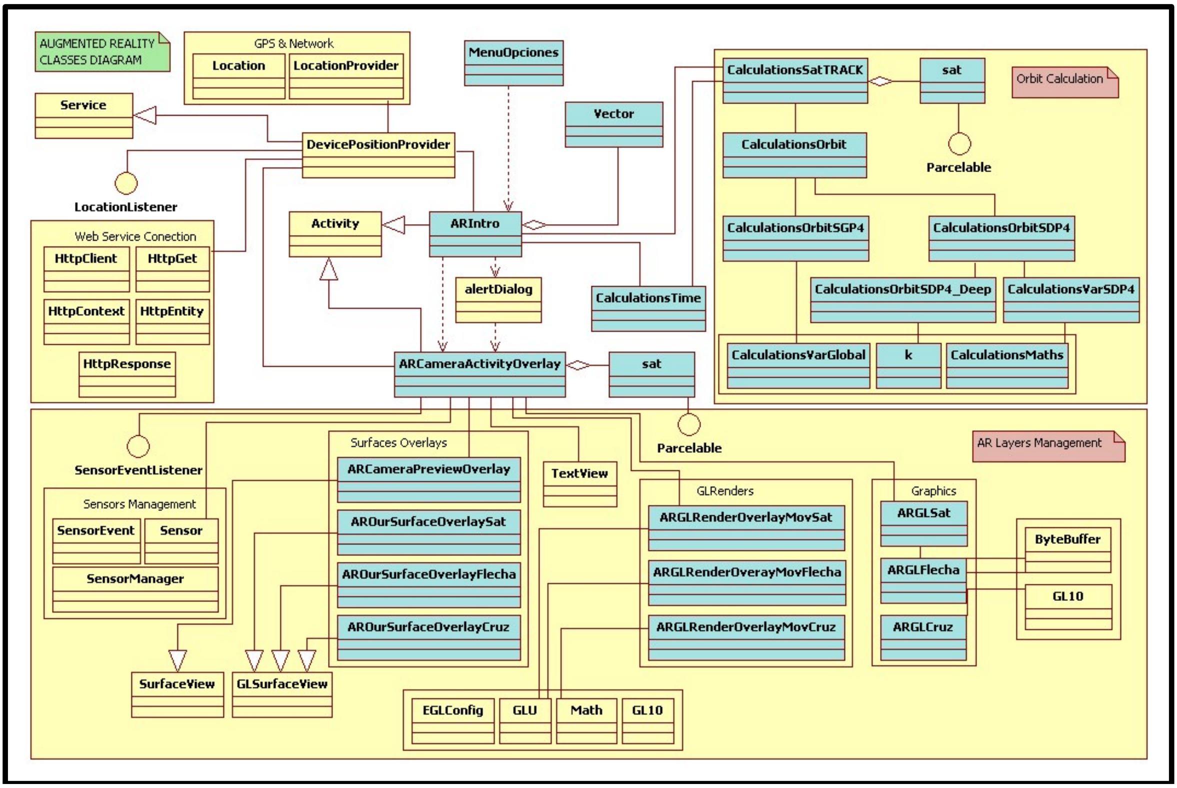


Figure 12. Augmented Reality Classes Diagram

Once the satellite has been picked, the AR view can be selected. Figure 12
Firstly `ARIntro` class:

- Runs the calculations from NORAD propagations orbit, with all the complex process of predict the orbit for the satellite. This will be explained in later sections.
- Also the device georeference parameters are retrieved with class `devicePoitionProvider`. GPS and network are used. Web service may be also needed.
- In case that the satellite elevation is less than 0, the satellites is under the horizon. Then an `alertDialog` is shown with information of the next overhead pass..

Then `ARCameraActivityOverlay` class:

- Manages the device orientation through Sensors management.
- Implements the Surface layers of the AR view: “Flecha”, “Cruz”, “Sat” and “Camera”.
- Implements the `GLRenders` of these surface layers, taking advantage of OPENGL features.
- Implements the objects of shown in these layers.
- Implements `textViews` with information about satellite and device orientation.

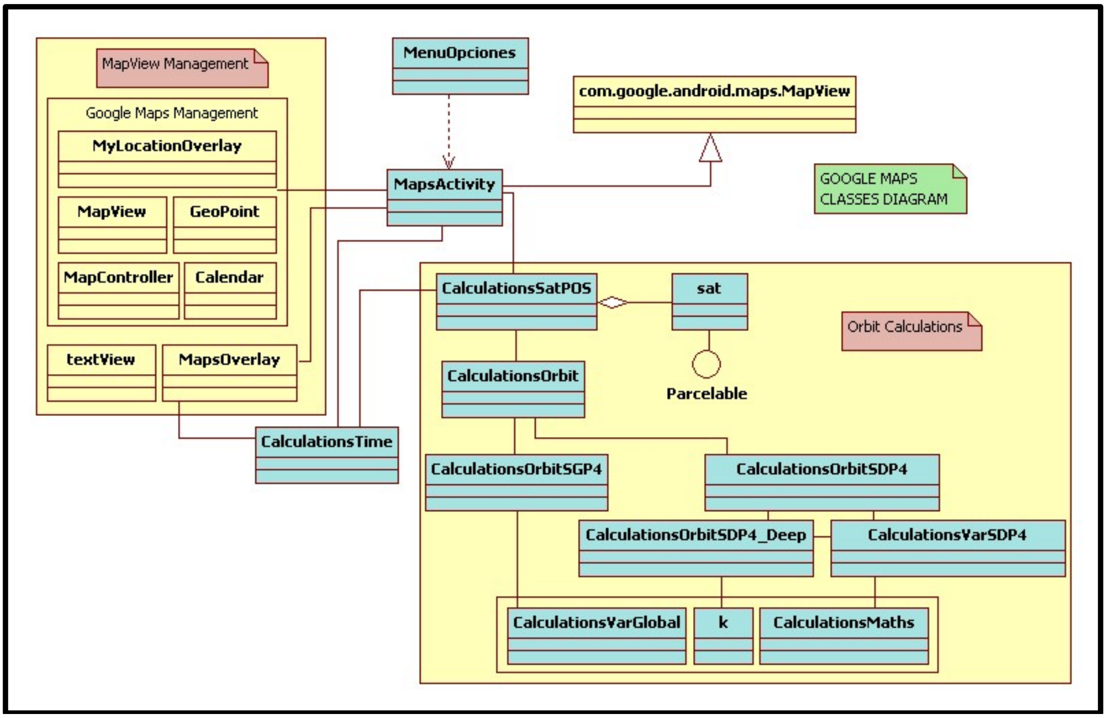


Figure 13. Google Maps Classes Diagram

The other functionality, locating the satellite in Google Maps:

- Creates the `mapView` class inherited from the system. It allows to use GOOGLE MAPS.
- There are also some other classes managing the characteristics of the current map.
- `textView` that show satellite location coordinates overlaying the map
- And again all the calculations where latitude, longitude and altitude of the satellite.

6.5.2. CLASSES DESCRITPION

Object	**`intro.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Welcome Screen, checks internet and shows logo		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`onCreate`	Bundle	-	Starts ZatDroid Welcome screen with logo Checks internet availability. Opens next Activity after 1 sec: `ListaTipoSat`
`isInternetOn`	-	Boolean	`ConnectivityManager` checks internet availability
`showAlert`	Title 2 messages	-	show an `alertDialog`
Superclasses	`Activity, ConnectivityManager, AlertDialog, Intent`		
Attributes	Thread timer		

Table 38. intro.java

Object	`listaTipoSat.java`		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Initial Menu that gives two options to search a sat: by type or by key words (name)		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`onCreate`	Bundle	-	Set Layout
`layoutCode`	width, height	LinearLayout	Creates adjustable Layout Components
`iconsLine`	2 Buttons width, height, 2 Drawable 2 Strings 2 int	LinearLayout	Defines one line with two icons in a linearLayout
`breadCrumbs`	width, height	LinearLayout	Sets *BreadCrumbs* buttons at the top of the screen
`scaleButtnTxt`	width, height p, text	XScale	calculate text width size to fit into box
`onClick`	View	-	Stores `SharedPreferences` “`tipoSatGeneral`” according to user selection starts `listaTipoSat2` or `searchActivity` Activity
`onBAckPressed`	-	-	finishes activity
Superclasses	`Activity, View.onClickListener, LinearLayout, ScrollView, Drawable, Button, Paint, SharedPreferences, Intent`		
Attributes	`Buttons (bSpecial, bWeather, bCommunications, bNavigation, bScientific, bMisc, bBreadCrumbs, bBreadCrumbsEnd), String (“MyFile”)`		

Table 39. listaTipoSat.java

Object	`searchActivity.java`		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	search by key words (name) activity manager		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`onCreate`	Bundle	-	Set Layout Handles query Intent
`onListItemClick`	listView,View position, id	-	Stores `sharedPreferences` “`SatElegido`” & “`tipoSatEspecifico`” of the sat picked runs `satElegidoSAXHandler` to search into XML for date from sat picked Creates “sat” Object with TLE data Calls `MenuOpciones.java`
`handleIntent`	intent	-	starts Android search interface calls `doSearch` method to perform search
`doSearch`	query	-	starts ProgressBar executes `Asynctask` “`MiTarea` calls “`buscarSatsEnXML`” and saves names in a list if second searches are retaken without download again txt

`scaleButtonText`	width, height p, text	XScale	calculate text width size to fit into box
`breadCrumbs`	width, height	LinearLayout	Sets *BreadCrumbs* buttons at the top of the screen
`breadCrumbsButton`	format variables	Button	sets format of *breadCrumbs* buttons
`mostrarAlerta`	Title 2 messages	-	show an `alertDialog`
`buscarSatsEnXML`	-	ArrayList<String>	Calls `searchListaSatsSAXHandler`. Using SAX parses XML and retrieves all sat names that fit query String (key words)
`loadUrlTLE`	-	ArrayList<String>	loads all URL from `celestrack` with txt files
`headerXML`	BufferWriter	-	writes the header of the XML file
`footerXML`	BufferWriter	-	writes the footer of the XML file
`writeXML`	BufferWriter BufferReader	-	reads txt files from URL in `celestrack.com` parses txt and writes XML file
`writeXSD`	BufferWriter	-	writes XSD Sheme
`MiTarea (`*`CLASS`*`)`	`Asyntask` that performs in background txt download and XML file creation. XML is created with all sats from all TLEs txt files. calls "`buscarSatsEnXML`" and saves names for the `listView`		
Superclasses	`listActivity, SAXParserFactory, SAXParser, XMLReader, ProgressDialog, Resources, LinearLayout, ScrollView, Button, BufferWriter, BufferREader, SharedPreferences, Intent`		
Attributes	`ArrayList<String> (sUrlTLE, alistaSat), String ("MyFile", "TLE_all.xml", "XSD_all.xml", query), BufferWriter (xml, xsd), BufferReader (in), sat (sat)`		

Table 40. searchActivity.java

Object	**`searchListaSatsSAXHandler.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Performs a search into the XML and find sat names that fits the string key words		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`getData`	-	ArrayList<String>	retrieves array with list of sat names found
`searchListaSatsHandler`	String	-	Constructor: it takes the query string
`parse`	InputStream Context	-	Performs the query. Search every node
Superclasses	`RootElement, Element, defaultHandler`		
Attributes	`ArrayList<String> (listaNombres), String (searchString)`		

Table 41. searchListaSatsSAXHandler.java

Object	**listaTipoSat2.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Second menu of satellite types		
Methods			
Name	**Inputs**	**Outputs**	**Description**
onCreate	Bundle	-	Set Layout Retrieves SahredPrefereces "TipoSatGeneral"
layoutCode	width, height typeSelected	LinearLayout	Creates adjustable Layout Components Selects the subtypes according to input type
iconsLine	button format param.	LinearLayout	Defines one line with two icons in a linearLayout
breadCrumbs	width, height typeSelected	LinearLayout	Sets *BreadCrumbs* buttons at the top of the screen
scaleButtonText	width, height p, text	XScale	calculate text width size to fit into box
onClick	View	-	calls saveData and saveData_openList
saveData	iTipoGuardado View	int	retrieves tipoEspecifico: subtype picked
savaData_openList	elegido	-	stores tipoEspecifico SharedPreferences opens listaSats.java
Superclasses	Activity, View.onClickListener, LinearLayout, ScrollView, Drawable, Button, Paint, SharedPreferences, Intent		
Attributes	Buttons (bLast30, bSpaceStations, b100Brightest, bBreadCrumbs, bBreadCrumbs2, bBreadCrumbsEnd), String ("MyFile", sNombrelistaelegida), ArrayString<String>		

Table 42. listaTipoSat2.java

Object	**listaSats.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	search by type activity manager and shows name list		
Methods			
Name	**Inputs**	**Outputs**	**Description**
onCreate	Bundle	-	Retrieves tipoEspecifico from SharedPref Set Layout calls seleccionarUrlTLE Creates Progress bar & Executes MiTarea
onListItemClick	listView,View position, id	-	Stores sharedPreferences "SatElegido" runs satElegidoSAXHandler to search into XML for date from sat picked Creates a "sat" Object with data from TLE Calls MenuOpciones.java

`scaleButtonText`	width, height p, text	XScale	calculate text width size to fit into box
`breadCrumbs`	width, height iSpecificType	LinearLayout	Sets *BreadCrumbs* buttons at the top of the screen according to iSepecificType
`breadCrumbsButton`	format variables	Button	sets format of breadCrumbs buttons
`mostrarAlerta`	Title 2 messages	-	show an `alertDialog`
`setButtonNames`	iType, select	String	returns SepecificType or GeneralType to set names of the icons
`buscarSatsEnXML`	-	ArrayList<String>	Calls `listaSatsSAXHandler`. Using SAX parses XML and retrieves all sat names.
`seleccionarUrlTLE`	type	String	loads URL from `celestrack` with txt files according to type
`escribirXML`	BufferWriter BufferReader	-	reads txt files from URL in `celestrack.com` parses txt and writes XML file
`escribirXSD`	BufferWriter	-	writes XSD Sheme
`MiTarea (CLASS)`	`Asyntask` that performs in background txt download and XML file creation. XML is created with all sats from TLEs txt file. calls "`buscarSatsEnXML`" and saves names for the `listView`		
Superclasses	`listActivity, SAXParserFactory, SAXParser, XMLReader, ProgressDialog, Resources, LinearLayout, ScrollView, Button, BufferWriter, BufferREader, SharedPreferences, Intent`		
Attributes	`ArrayList<String> (alistaSat), String (sUrlTLE, "MyFile", "TLE.xml", "XSD.xml", query), sat (sat)`		

Table 43. listaSats.java

Object	**`ListaSatsSAXHandler.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Performs a search into the XML and find sat names		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`getData`	-	ArrayList<String>	retrieves array with list of sat names found
`parse`	InputStream Context	-	Performs the query. Search every node
Superclasses	`RootElement, Element, defaultHandler`		
Attributes	`ArrayList<String> (listaNombres)`		

Table 44. listaSatsSAXHandler.java

Object	**`satElegidoSAXHandler.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Performs a search into XML, find sat parameters, creates a sat object		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`getData`	-	sat	retrieves sat object with TLE parameters of the sat picked

`satElegidoSAXHandler`	String	-	Constructor: it takes the sat name
`parse`	InputStream Context	-	retrieves sateElegido from `SharePref` Performs the query. Search every node
Superclasses	`RootElement, Element, defaultHandler`		
Attributes	`String (satElegido)`		

Table 45. satElegidoSAXHandler.java

Object	**`sat.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Object with all data of the satellite picked		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`sat`	-	-	Constructor
`sat(…)`	parameters	-	Constructor: it takes the parameters
`get…`	-	attribute	retrieves a certain attribute
`set…`	value	-	sets a certain attribute value
`sat(…)`	Parcel	-	parcelling part
`copyData`	sat	-	copies data to another sat object
`showLog`	-	-	writes object content to logCat
`writeToparcel`	Parcel, int	-	parcelling part
Superclasses	`Parcelable`		
Attributes	`name, number, inclination, raan, eccentricity, argumentPerigee, meanAnomly, meanMotion, epochYear, epoch, epochC, epochN, epochJD, meanMotionDerivate, bstarDragTerm, epehemeridesType, encontrad, r1, r2, r3, rdot1, rdot2, rdot3, lat, lon, az, ele, alt, range`		

Table 46. sat.java

Object	**`menuOpciones.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Menu with the two functionalities		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`onCreate`	Bundle	-	Set Layout Retrieves `SahredPref`"TipoSatEspecifico"
`indentificarBotones`	-	-	Identify buttons layout
`layoutCode`	width, height typeSelected	LinearLayout	Creates adjustable Layout Components Selects subtypes according to input type
`iconsLine`	button format param.	LinearLayout	Defines one line with two icons in a linearLayout
`breadCrumbs`	width, height iSpecificType	LinearLayout	Sets *BreadCrumbs* buttons at the top
`breadCrumbs2`	width, height	LinearLayout	Sets *BreadCrumbs* 2nd line with name
`breadCrumbsButton`	format variables	Button	sets format of *breadCrumbs* buttons
`setButtonNames`	iType, select	String	returns SepecificType or GenralType to set names of the icons

scaleButtonText	width, height p, text	XScale	calculate text width size to fit into box
onClick	View	-	calls ARintro when AR button is pressed (previous ProgressDialog created) calls MapActivity if Map button clicked.
Superclasses	Activity, View.onClickListener, LinearLayout, ProgressDialog, SharedPreferences, Intent		
Attributes	Buttons (R, Map), sat, ProgresDialog		

Table 47. menuOpciones.java

Object	**MapsActivity.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Google Maps functionality		
Methods			
Name	**Inputs**	**Outputs**	**Description**
onCreate	Bundle	-	Set Layout Retrieves at object calculates current time calls calculationsSatPOS.class to calculate position creates layer list ,add compass and MapOverlay class
Superclasses	MapActivity, MyLocationOverlay, MapView, MapsController, Dundle, Calendar, GeoPoint, MapOverlay		
Attributes	MyLocationOverlay (Compass), MapView (map)		

Table 48. MapsActivity.java

Object	**MapsOverlay.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Google Maps layer class		
Methods			
Name	**Inputs**	**Outputs**	**Description**
MapOverlay	sat	-	Constructor: set current time calls method trajectory for the first time create Geopoint with sat position
trajectory	sat	double[]	Calculates sat trajectory calling CalculationsSatPOS 20 positions predicted (1 per minute, 10 past, 10 future) & creates array
draw	Canvas, MapView, boolean	Boolean log	refreshes view instantly updates trajectory (1 second delay) draws sat icon and updates its geoPoint draw text info box
Superclasses	com.google.android.maps.Overlay, Calendar		
Attributes	CalculationsTime, sat, double[] (trajectpoints)		

Table 49. MapsOverlay.java

Object	**ARIntro.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	First Augmented reality manager		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`onCreate`	Bundle	-	retrieves sat object calls `DevicePositionProvider` for sat position: lat, lon, alt calls `CalculationsSatTRACK` for sat orientation (azimuth, elevation) if ele<0, calculate next overhead pas and show message calls `ARCameraActivityOverlay`
`showAlert`	Title 2 messages	-	show an `alertDialog`
Superclasses	`Activity, Calendar, AlertDialog`		
Attributes	`double (latitude, longitude, altitude), sat`		

Table 50. ARIntro.java

Object	**ARCameraActivityOverlay.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR View Manager		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`onCreate`	Bundle	-	Create Camera, Sensors, renders, `surfaceViews` and two `TextViews` Set Layout with previous layers
`getCameraInstance`	-	Camera	starts camera
`onSensorChanged`	SensorEvent	-	refreshed Sensors values `getRotationMatrix` and Orientation Vector from device Calculates device azimuth and elevation and send them to the renders update `textView` with device info
`showAlert`	Title 2 messages	-	show an `alertDialog`
`onStop`	-	-	unregisters Sensor Listeners
Superclasses	`SensorManager, Sensor, FrameLayout, Camera, SensorEvent, AlertDialog`		
Attributes	`Camera, ARCameraPreview_Overlay, AROurSurfaceOverlaySat, AROurSurfaceOverlayCruz, AROurSurfaceOverlayFlecha, SensorManager, Sensor, float[] (several), DevicePositionProvider, ARGLRenderOverlayMovSat, ARGLRenderOverlayMovCruz, ARGLRenderOverlayMovFlecha`		

Table 51. ARCameraActivityOverlay.java

Object	**ARCameraPreview_Overlay.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR Camera SurfaceView		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`ARCameraPreview_Overlay`	Camera Context	-	Constructor: sets surfaceholder
`surfaceCreated`	SurfaceHolder	-	sets Camera
`surfaceChanged`	SurfaceHolder int	-	starts and configure camera
Superclasses	SurfaceView, SurfaceHolder, Camera		
Attributes	Camera, SurfaceHolder		

Table 52. ARCameraPreview_Overlay.java

Object	**AROurSurfaceOverlaySat.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR sat GLSurfaceView		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`AROurSurfaceOverlaySat`	Render Context	-	Constructor: Sets render and configure Surface properties. Transparent & setZOrderOnTop
Superclasses	GLSurfaceView, Context		
Attributes	-		

Table 53. AROurSurfaceOverlaySat.java

Object	**AROurSurfaceOverlayCruz.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR central square GLSurfaceView		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`AROurSurfaceOverlayCruz`	Render Context	-	Constructor: Sets render and configure Surface properties Transparent & setZOrderOnTop
Superclasses	GLSurfaceView, Context		

Table 54. AROurSurfaceOverlayCruz.java

Object	**AROurSurfaceOverlayFlecha.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR arrow GLSurfaceView		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`AROurSurfaceOverlayFlecha`	Render Context	-	Constructor: Sets render and configure Surface properties Transparent and setZOrderOnTop
Superclasses	GLSurfaceView, Context		
Attributes	-		

Table 55. AROurSurfaceOverlayFlecha.java

Object	**ARGLRenderOverlayMovSat.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR Sat Render		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`receiveOrientation`	elevation azimuth	-	receives devic orientation & calls `filterSmoothMovements`(values)
`filterSmoothMovements`	double double	double	Smooths values of elevation and azimuth not to let sat icon vibrate in AR Camera view
`receiveGravityVector`	float[]	-	Receives gravity vector from `ARCameraActivityOverlay`
`ARGLRenderOverlayMovSat`	Context, sat	-	Constructor
`onSurfaceCreated`	GL10 EGLConfig	-	Configures Surface
`onDrawFrame`	GL10	-	Configures and updates OPENGL camera, screen, sat icon and 3D geometry
`onSurfaceChanged`	Gl10, int	-	updates surface configuration
Superclasses	`GL10, EGLConfig, Renderer, Context`		
Attributes	`ARGLSat, sat, double (eleSat, azSat, dist, eleDevice, azDevice)`		

Table 55a. ARGLRenderOverlayMovSat.java

Object	**ARGLRenderOverlayMovCruz.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR central square Render		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`receiveOrientation`	elevation azimuth	-	receives devic orientation & calls `filterSmoothMovements`(values)
`ARGLRenderOverlayMovCruz`	Context, sat	-	Constructor
`onSurfaceCreated`	GL10 EGLConfig	-	Configures Surface
`onDrawFrame`	GL10	-	Configures and updates OPENGL camera, screen, sat icon and 3D geometry
`onSurfaceChanged`	Gl10, int	-	updates surface configuration
Superclasses	`GL10, EGLConfig, Renderer, Context`		
Attributes	`ARGLCruz, sat, double (eleSat, azSat, dist, eleDevice, azDevice)`		

Table 56. ARGLRenderOverlayMovCruz.java

Object	**ARGLRenderOverlayMovFlecha.java**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR arrow Render		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`receiveOrientation`	elevation azimuth	-	receives device orientation & calls `filterSmoothMovements`(values)

`filterSmoothMovements`	double double	double	Smooths values of elevation and azimuth not to let sat icon vibrate in AR Camera view
`ARGLRenderOverlayMovFlecha`	Context, sat	-	Constructor
`onSurfaceCreated`	GL10 EGLConfig	-	Configures Surface
`onDrawFrame`	GL10	-	Configures and updates OPENGL camera, screen, sat icon and 3D geometry
`onSurfaceChanged`	Gl10, int	-	updates surface configuration
Superclasses	`GL10, EGLConfig, Renderer, Context`		
Attributes	`ARGLFlecha, sat, double (eleSat, azSat, dist, eleDevice, azDevice)`		

Table 57. ARGLRenderOverlayMovFlecha.java

Object	**`ARGLSat.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR Sat Object		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`loadGLTexture`	GL10 Context	-	configures texture
`Update`	float[], float[]	-	updates square geometry & icon
`ARGLSat`	-	-	Constructor
`draw`	GL10	-	refreshing parameters
Superclasses	`FloatBuffer, ShortBuffer, GL10, Context, InputStream, Bitmap, GLUtils, BytrBuffer`		
Attributes	`FloatBuffer, ShortBuffer`		

Table 58. ARGLSat.java

Object	**`ARGLCruz.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR central square Object		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`Update`	float[], float[]	-	-
`ARGLCruz`	-	-	Constructor: updates square geometry and icon
`draw`	GL10	-	refreshing parameters
Superclasses	`FloatBuffer, ShortBuffer, GL10, Context, InputStream, Bitmap, GLUtils, BytrBuffer`		
Attributes	`FloatBuffer, ShortBuffer`		

Table 59. ARGLCruz.java

Object	ARGLFlecha.java		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	AR arrow Object		
Methods			
Name	Inputs	Outputs	Description
loadGLTexture	GL10 Context	-	configures texture
Update	float[], float[]	-	updates square geometry and icon
ARGLFlecha	-	-	Constructor
draw	GL10	-	refreshing parameters
Superclasses	FloatBuffer, ShortBuffer, GL10, Context, InputStream, Bitmap, GLUtils, BytrBuffer		
Attributes	FloatBuffer, ShortBuffer		

Table 60. ARGLFlecha.java

Object	vector.java		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Mathematical vector operations		
Methods			
Name	Inputs	Outputs	Description
vector	double, double double	-	constructor
vector	vector	-	constructor
set…	values	-	sets values
get…	-	values	gets values
clone	-	object	clones
getCoords	-	double[]	gets coordinates
equals	object	boolean	compares object
add	vector	vector	-
sustract	vector	vector	-
multiply	double	vector	-
crossProduct	vector	vector	-
dotProduct	vector	double	-
magnitude	-	double	-
normalize	-	vector	-
angle	vector	double	-
toString	-	String	-
Superclasses	-		
Attributes	doubles (v1, v2, v3)		

Table 61. Vector.java

Object	`DevicePostitionProvider.java`		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Provides latitude, longitude and altitude of device		
Methods			
Name	Inputs	Outputs	Description
`devicePositionProvider`	Context	-	constructor
`getLocation`	-	Location	Retrieves location from network and GPS For altitude, GPS or web service is used
`stopUsingGPS`	-	-	-
`get…`	-	variable	gets value
`showSettingsAlert`	-	-	shows `alertDialog` to change settings
`getElevationFromGoogleMaps`	longitude, latitude	double	connects to web service GOOGLE MAPS Elevation to retrieve altitude from location
Superclasses	`Service, Location, LocationListener, LocationManager, AlertDialog, Intent, HttpClient, HttpContext, HttpGet, HttpResponse, HttpEntity, InputStream`		
Attributes	`Context, Booleans, Location, double (latitude, longitude, altitude), LocationManager`		

Table 62. DevicePositionProvider.java

Object	`CalculationsSatPOS.java`		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Calculates satellite latitude, longitude and altitude.		
Methods			
Name	Inputs	Outputs	Description
`satPOS`	currentTime sat	sat	Calls `CalculationsOrbit` for position and velocity vectors calls `latlon` method for latitude longitude and altitude
`latlon`	currentTime sat	sat	It calculates geodetic latitude, longitude and altitude of satellite.
Superclasses	-		
Attributes	`CalculationsTime`, `CalculationsOrbit`		

Table 63. CalcualtionsSatPOS.java

Object	`CalculationsSatTRACK.java`
Version	V1.0 – 15/08/2013
Author	Rodrigo Santos Álvarez
Description	Calculates satellite azimuth and elevation from device location

Methods			
Name	**Inputs**	**Outputs**	**Description**
`satTRACK`	currentTime sat DevicePosition	sat	Calls `CalculationsOrbit` for position and velocity vectors calls “`geo2eciobs`” method for device position in ECI calls “`aziele`” method for sat azimuth and elevation
`geo2eciobs`	currentTime devicePosition	vector	calculates device position in ECI
`aziele`	sat, r, ground, deviceposition, currentTime	sat	calculates sat azimuth and elevation and returns sat object completed
Superclasses	-		
Attributes	`CalculationsTime`, `CalculationsOrbit`		

Table 64. CalcualtionsSatTRACK.java

Object	**`CalculationsOrbit.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Manages calculations SGP4/SDP4: position and velocity vectors		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`satPrepareTimeData`	sat	sat	configures time epochN-epochJD
`prepareTLEdata`	sat	CalculationsarGlobal	configures TLE data
`orbit`	tsinceEpoch sat	sat	According to result from select_`ephemeris` method orbit is calculated with `CalculationOrbitSDP4` or `CalculationsOrbitSGP4`
`select_ephemris`	CalculationsVarglobal	int	select SGP4 or SDP4 calculations methods
Superclasses	-		
Attributes	-		

Table 65. CalcualtionsOrbit.java

Object	**`CalculationsOrbitSGP4.java`**		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	implements SGP4 NORAD method		
Methods			
Name	**Inputs**	**Outputs**	**Description**
`resetStaticVars`	-	-	configures variables
`getOrbit`	tsince, sat CalculationsVarGlobal	sat	calculates position and velocity with SGP4
Superclasses	-		
Attributes	-		

Table 66. CalcualtionsOrbitSGP4.java

Object	CalculationsOrbitSDP4.java		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	implements SDP4 NORAD method		
Methods			
Name	Inputs	Outputs	Description
resetStaticVars	-	-	configures variables
getOrbit	tsince, sat CalculationsVarGlobal	sat	calculates position and velocity with SDP4. It uses CalculationsVarSDP4, CalculationsOrbitSDP4_Deep
Superclasses	-		
Attributes	-		

Table 67. CalcualtionsOrbitSDP4.java

Object	CalculationsOrbitSDP4_Deep.java		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	implements SDP4 NORAD method variables		
Methods			
Name	Inputs	Outputs	Description
deepCalc	int, Vriables	Variables SDP4	configures variables SDP4
Superclasses	-		
Attributes	-		

Table 68. CalcualtionsOrbitSDP4_Deep.java

Object	CalculationsVarSDP4.java		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Variables used in SDP4		
Methods			
Name	Inputs	Outputs	Description
get…	-	value	get values from variable
set…	value	-	set value for a variable
copyData	CalculationsVarSDP4	-	copy object
showlog	-	-	show content in logCat
Superclasses	-		
Attributes	SDP4 variables		

Table 69. CalcualtionsVarSDP4.java

Object	CalculationVarGlobal.java		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Variables used in SDP4 and SGP4		
Methods			
Name	Inputs	Outputs	Description
get…	-	value	get values from variable
set…	value	-	set value for a variable
copyData	CalculationsVarGlobal	-	copy object
showlog	-	-	show content in logCat
Superclasses	-		
Attributes	SDP4 and SGP4 variables		

Table 70. CalcualtionsVarGlobal.java

Object	`CalculationsTime.java`		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Methods used to transform time formats		
Methods			
Name	Inputs	Outputs	Description
`julian`	year, month, day	double	conventional to julian date
`Julian_date_of_year`	year	doble	Julian date of year
`Julia_date_of_epoch`	epoch, epochYear	double	Julian date of (epoch in NORAD format)
`DOY`	year, month, day	int	day of year
`Fraction_of_day`	hour, min, sec	double	fraction of a day
`Julian_date`	Calendar	double	Calendar type to Julian date
`getCalendarFromJD`	jd	Calendar	julian Date to Calendar
`ThetaG_JD`	double	double	Greenwich mean sidereal time from Julian date
`thetaG`	epoch, epochYear	double	Greenwich mean sidereal time from NORAD date
`sat_Eclipsed`	vector, vector, depth	int	calculates sat eclipse status and depth
`test`	-	int	carries out tests of every method
Attributes	CalculationMaths		

Table 71. CalcualtionsTime.java

Object	`CalculationMaths.java`		
Version	V1.0 – 15/08/2013		
Author	Rodrigo Santos Álvarez		
Description	Mathematical functions		
Methods			
Name	Inputs	Outputs	Description
`FMod2p`	double	double	calculates 0-2 π angle
`FixAngle`	double	double	calculates 0-2 π angle
`Frac`	double	double	returns fractional part
`Round`	double	int	returns argument rounded to nearest integer
`Int`	double	double	floor integer
`Test`	-	int	carries out tests of every
Attributes	CalculationMaths		

Table 72. CalcualtionsMaths.java

Object	`k.java`
Version	V1.0 – 15/08/2013
Author	Rodrigo Santos Álvarez
Description	Mathematical constants

Table 73. k.java

Object	`visibleSats.java`
Description	Calculates list of visible sats (ele>0).
This class is not implemented. It lasts 35 minutes and it is out of requirements It gets a list of sat names that can be tracked (ele>0) from device location.	

Table 74. visibleSats.java

6.6. BEHAVIOR MODEL

6.6.1. USE CASES DIAGRAM

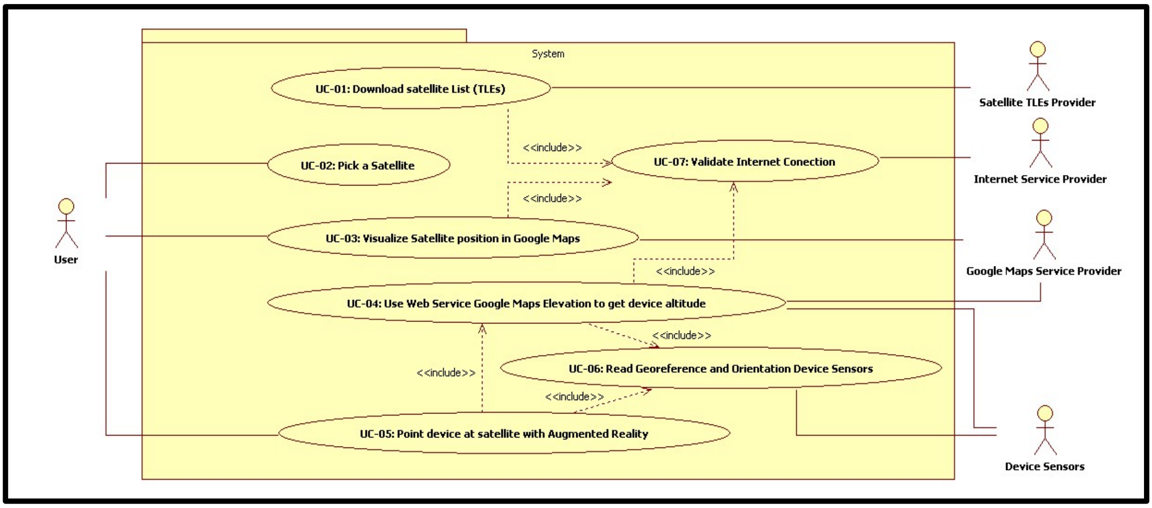


Figure 14. Use Cases Diagram

This diagram is repeated, but it is considered important to introduce next diagrams.

6.6.2.SEQUENCE DIAGRAMS

Only the most representative diagrams are shown.

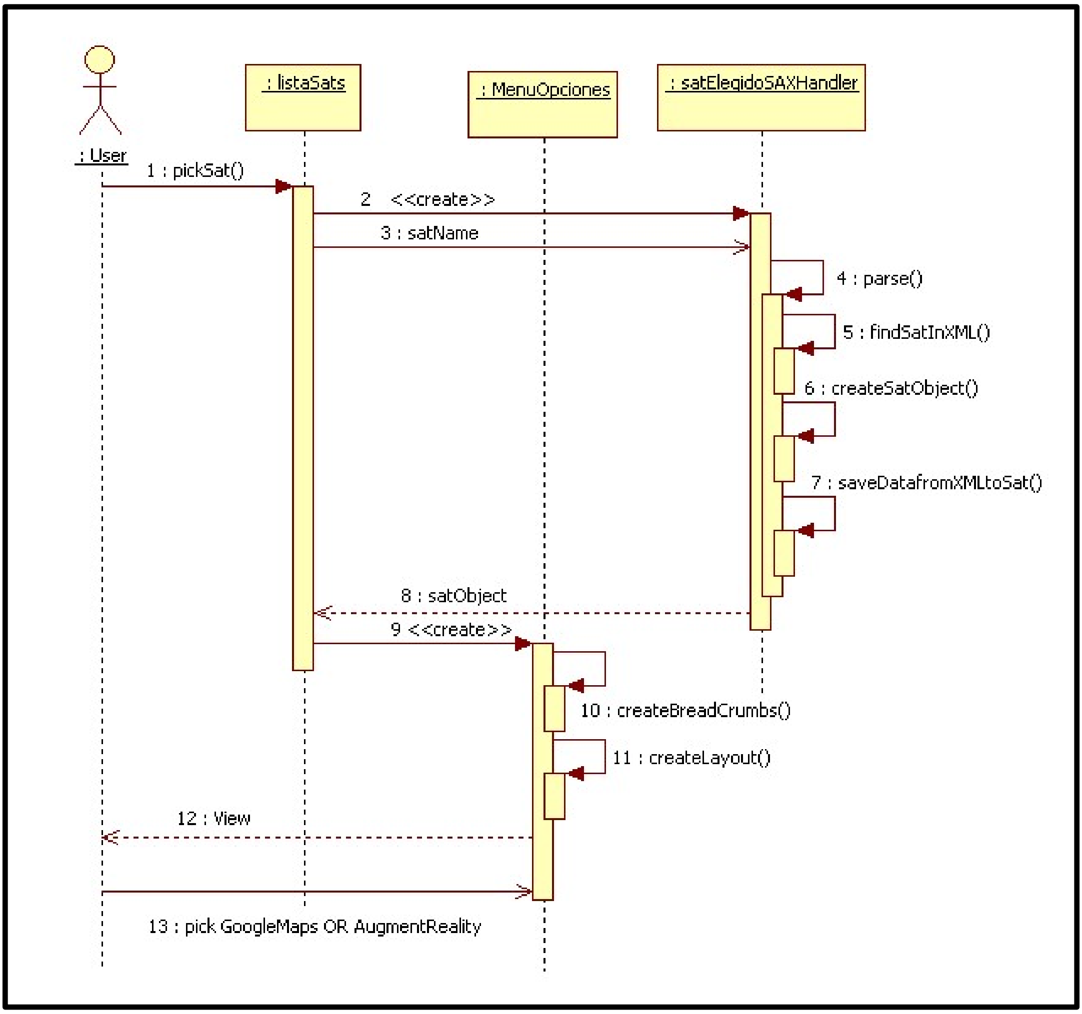


Figure 15. Sequence Diagram 01: Creating sat JAVA Object from XML

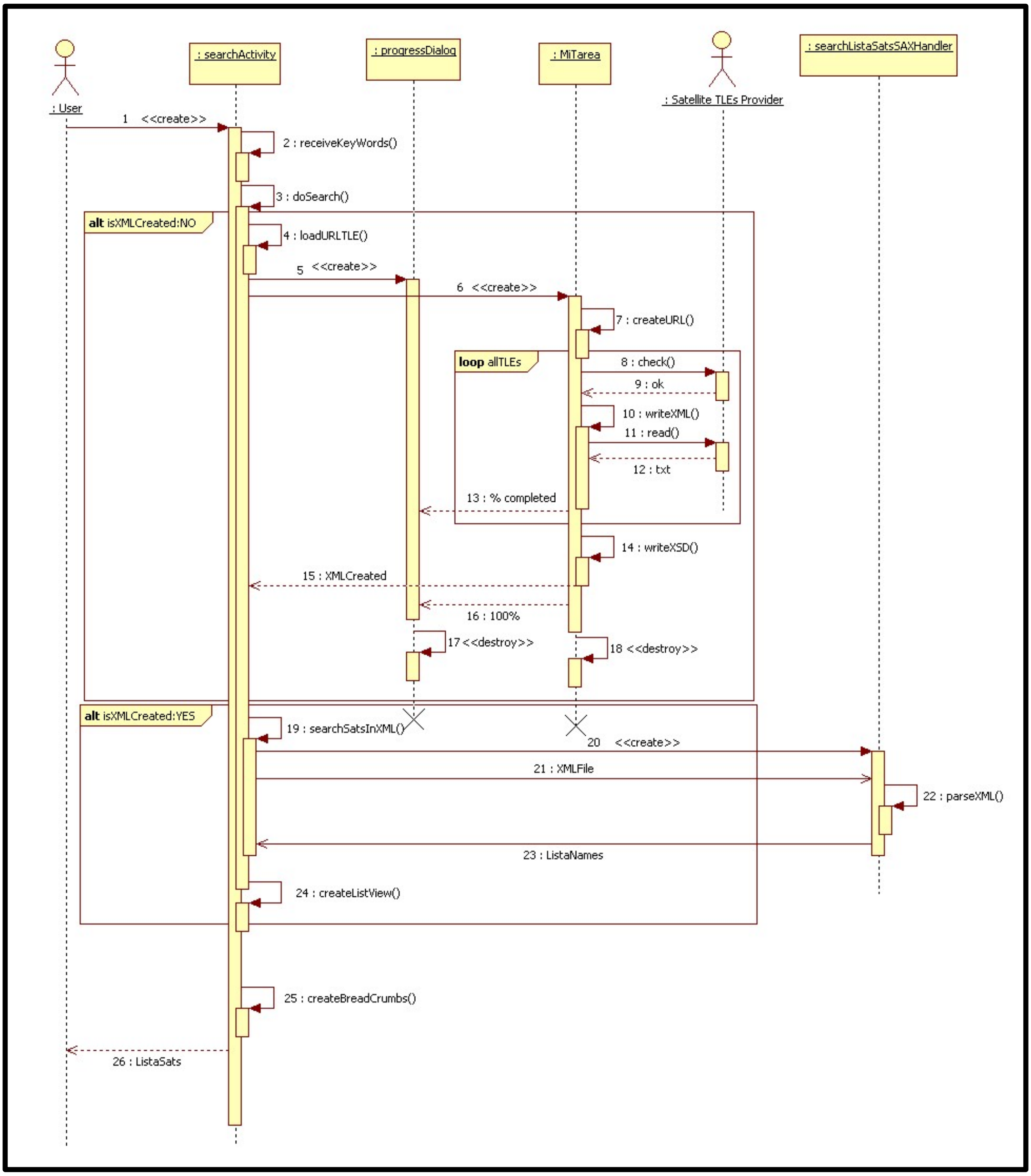


Figure 16. Sequence Diagram 02: UC-01: Download Satellite list (search by name)

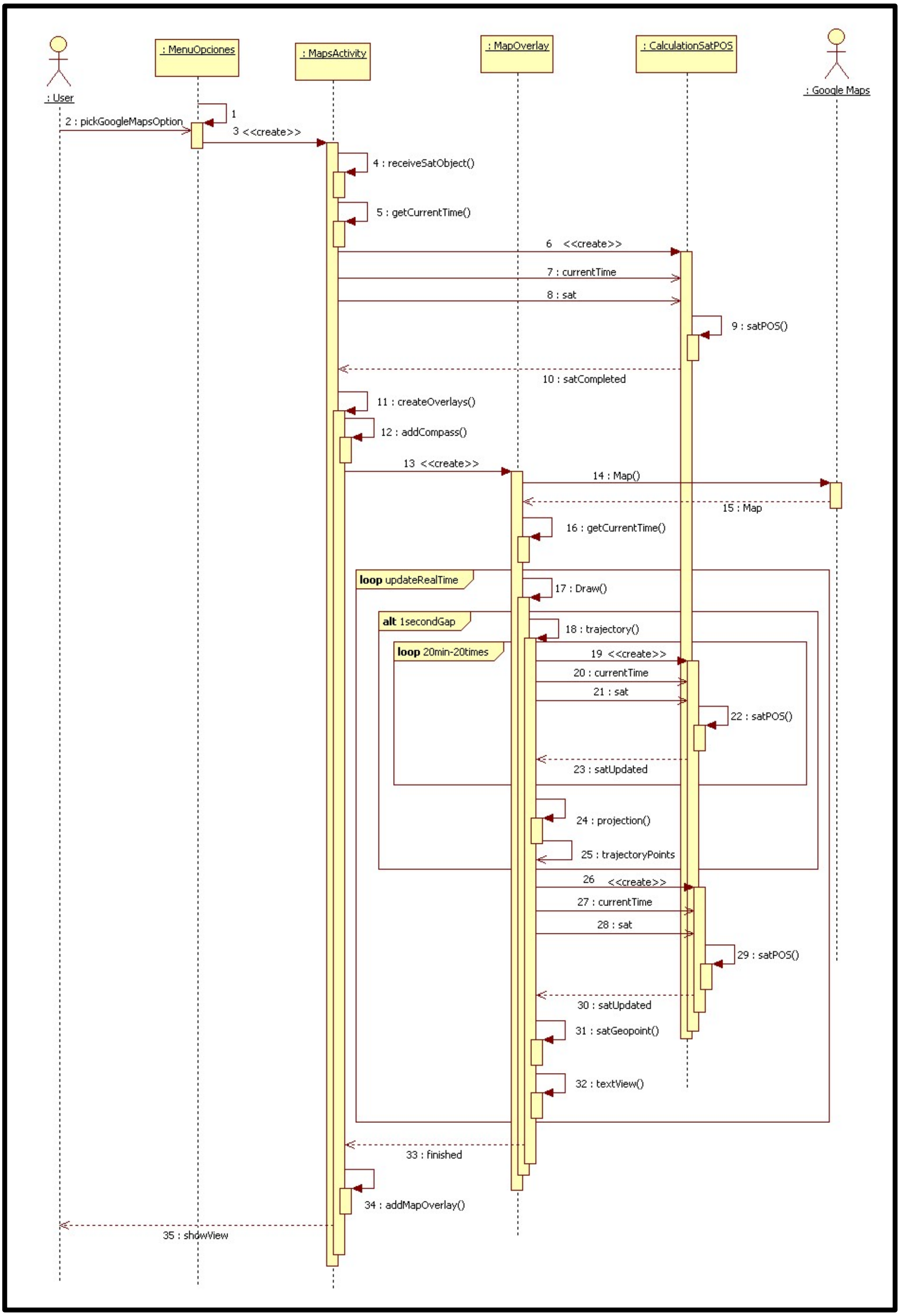


Figure 17. Sequence Diagram 03: UC-03: GOOGLE MAPS View

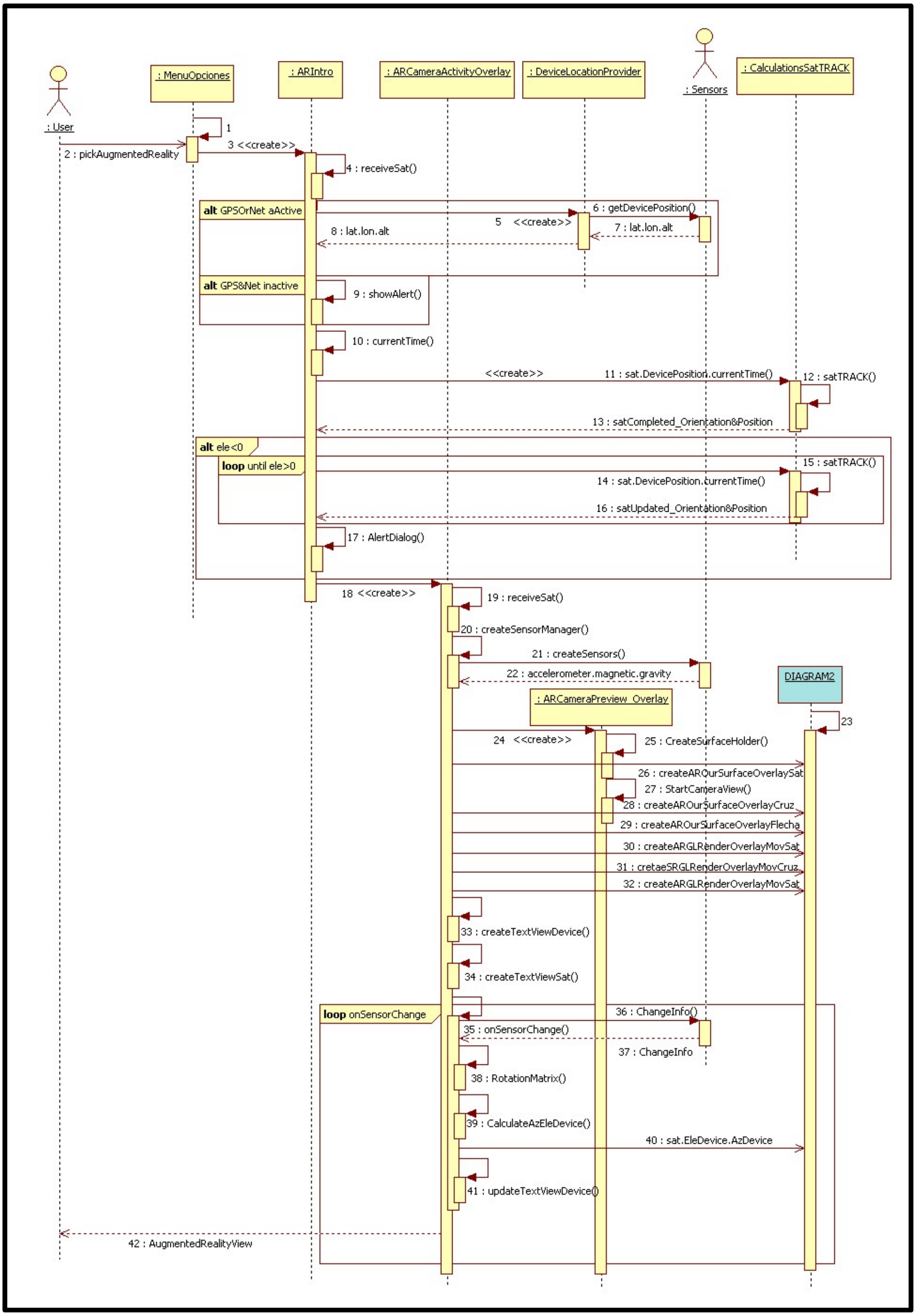


Figure 18. Sequence Diagram 04: UC-05: Augmented Reality View (I)

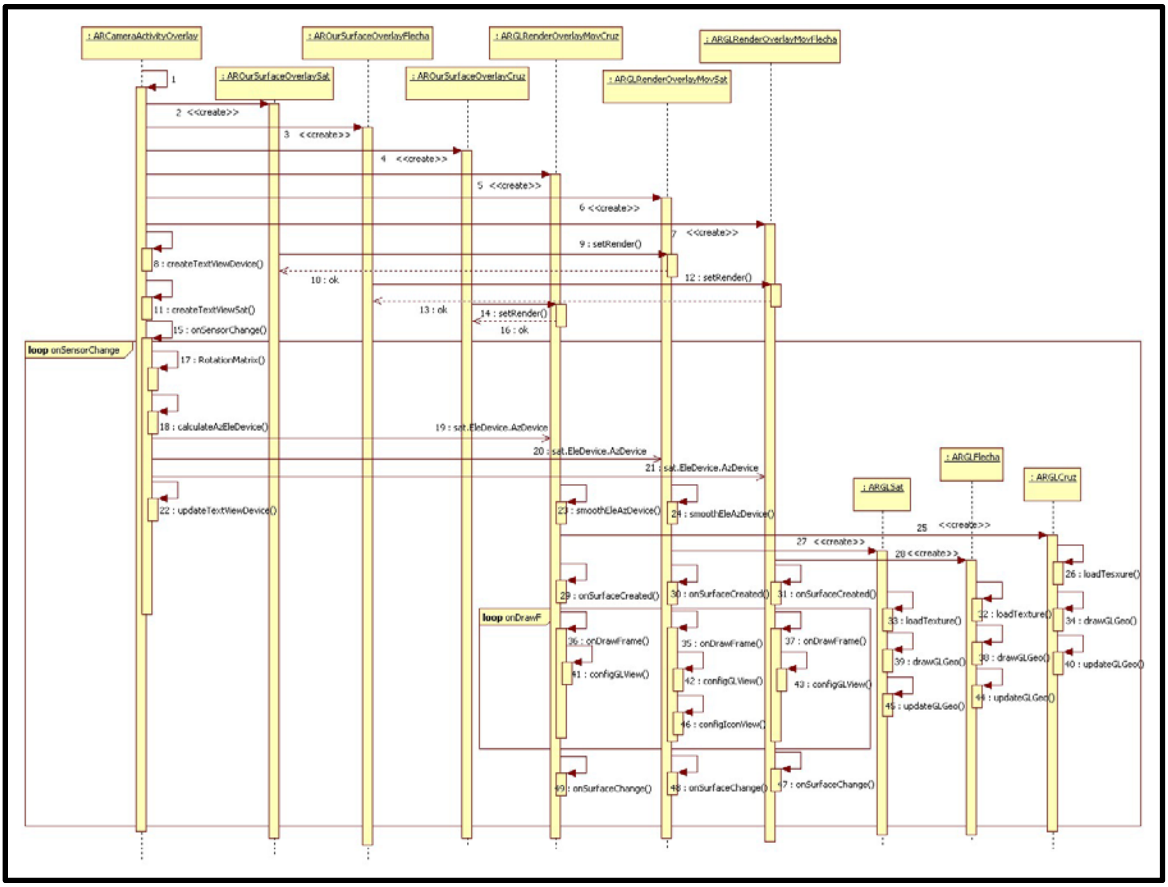


Figure 19. Sequence Diagram 05: UC-05: Augmented Reality View (II)

6.7. SUMMARY

CLASSES DIAGRAMS	General Classes Diagram
	Satellite Explorer Classes Diagram
	Augmented Reality classes Diagram
	GOOGLE MAPS Classes Diagram
OBJECT TYPE	`intro`
	`listaTipoSats`
	`searchActivity`
	`MiTarea`
	`searchListaSatsSAXHandler`
	`listaTipoSat2`
	`listaSats`
	`listaSatsSAXHandler`
	`satElegidoSAXHandler`
	`sat`
	`MenuOpciones`
	`MapsActivity`
	`MapOverlay`
	`ARIntro`
	`ARCameraActivityOverlay`
	`ARCameraPreview_Overlay`
	`AROurSurfaceOverlaySat`
	`AROurSurfaceOverlayFlecha`
	`ARSurfaceOverlayCruz`
	`ARGLRenderOverlayMovSat`
	`ARGLRenderOverlayMovCruz`
	`ARGLRenderOverlayMovFlecha`
	`ARGLSat`
	`ARGLFlecha`
	`ARGLCruz`
	`Vector`
	`DevicePostionProvider`
	`CalculationsSatPOS`
	`CalculationsSatTRACK`
	`CalculationsTime`
	`CalculationsMaths`
	`CalculationsOrbit`
	`CalculationsOrbitSDP4`
	`CalculationsOrbitSGP4`
	`CalculationsOrbitSDP4_Deep`
	`CalculationsVarGlobal`
	`CalculationsVarSDP4`
	`CalculationsMaths`
	`k`
	`visibleSats`
SEQUENCE DIAGRAM	Creating sat JAVA Object from XML
	UC-01: Download Satellite list (search by name)
	UC-03: GOOGLE MAPS View
	UC-05: Augmented Reality View (I)
	UC-05: Augmented Reality View (II)
DEPLOYMENT DIAGRAM	Deployment Diagram
PACKAGE DIAGRAM	Packages Diagram

Table 75. Design summary

7. USER INTERFACE DESIGN

7.1. GUI REQUIREMENTS

Once the design of the back-end has been determined, it is now time for the graphical user interface. As it is the door of the whole application to the user, it is really essential that it fits some requirements. There are brilliant apps in the market whose GUI is not as expected. Therefore, it is not successful. Moreover, nowadays GUI´s importance is increasing as we live in a visual world expecting immediate results. ZatDroid GUI is required to fit some expectations:

- Friendly, beautiful, attractive and colourful.
- Multilanguage support: English and Spanish. Automatically selected with default device language.
- Icon based: icons must be the way to navigate and must be representative.
- *BreadCrumbs* buttons: indicate the selections picked, where is the user within the app and allow navigating backwards
- `ListViews` for long lists
- Progress bars shown when a process is taking some time, e.g., downloading.
- Support different device screen sizes.
- Delay between screen must be less than 3 seconds.

7.2. SCENARIOS

In order to clearly describe the processes running under every screen, some scenarios are going to be defined. Although the class diagrams and descriptions has clarify the flow, it is considered really helpful to explain the flow following the user experience. Sequence diagrams could have been created, but an example seemed too long and complicated to understand, so a brief description is considered more appropriate.

These Scenarios will be defined taking into account Figure 20

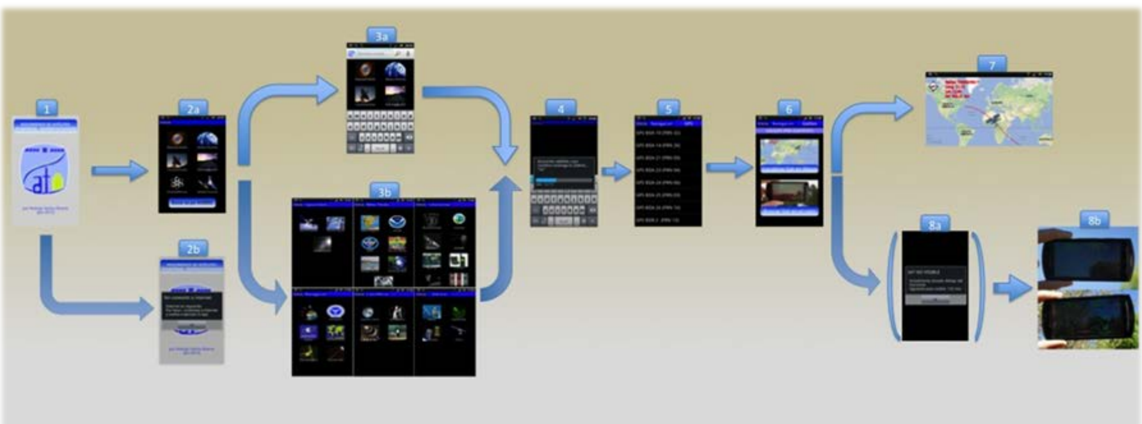


Figure 20. Scenarios

7.2.1. SCENARIO 1: INTRO

General Description: Welcome screen with icon, author and title.

Figure 21. Scenario 1: Intro

OnCreate:

- `Intro.java`
- Layout “`intro.xml`” in a thread waiting 1 second.
- `ConnectivityManager` check if there is internet available.
  - If it is not, go to Scenario 2b.
  - If it is, go to scenario 2a:

onClick: No clickable screen. It redirects automatically.

7.2.2. SCENARIO 2a: INITIAL SEARCH MENU

General Description: home menu with option to start satellite search

Figure 22. Scenario 2: Initial search menu

onCreate:

- Layout set with code to support different screen sizes, adjustable buttons and text.
- *BreadCrumbs* at the top. As it is home, there no navigation available.
- Text support English and Spanish, as shown in the images.

onClick:

- Any icon:
    - The type picked is saved in `SharedPreferences` as “*tipoSatGeneral*”
    - Go to Scenario 3b.

- “Search by name” button:
    - launch “`onSearchRequested`” and go to Scenario 3a.

7.2.3. SCENARIO 2b: INTERNET CONNECTION MESSAGE

General Description: `AlertDialog` about internet connexion availabitilty.

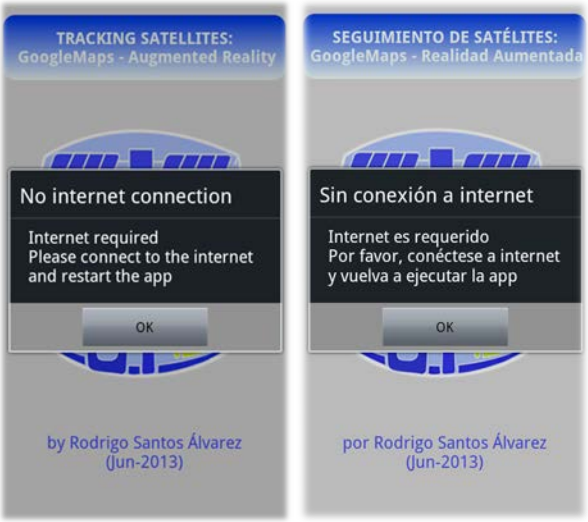


Figure 23. Scenario 2b: Internet connection message

onCreate:

- `AlerDialog` coming from scenario 1. Internet is not available.
- Supports English and Spanish (as shown in images)

onClick:

- OK button close the application. It needs Internet to perform as expected.

7.2.4. SCENARIO 3a: SEARCH BY KEY WORDS (NAME)

General Description: search by name Layout with ANDROID format.

Figure 24. Scenario 3a: search by key words (name)

onCreate:
Method `handleIntent` from `searchActivity.java` starts *SEARCH_ACTION* which shows the familiar ANDROID search view, with keyboard at the bottom and the box at the top. It also allows voice searches.

onClick:
After the user has written some key words to search a satellite by its name, clicks on the "search button". Then method "`doSearch`" from `searchActivity` is called with the query string as a parameter. Go to Scenario 4.

7.2.5. SCENARIO 3b: SEARCH BY TYPE

General Description: screens showing satellite subtypes.

Figure 25. Scenario 3b: search by type

onCreate:
`listaTipoSat2.java` is invoked.

- tipoSatGeneral value is retrieved from `SharedPreferences`. Depending on this value, one of this 6 screen is displayed matching the type picked in scenario 2a.
- Layout set with code to support different screen sizes, adjustable buttons and text.
- *BreadCrumps* at the top supporting English and Spanish.

onClick:

- *BreadCrumbs* first button: navigates backwards to scenario 2a again
- Icon: stored in `SharedPreferences` the value as “tipoSatEspecifico”. Go to Scenario 4.

7.2.6. SCENARIO 4: DOWNLOADING PROGRESS BAR

General Description: progress bar with information of the download process.

Figure 26. Scenario 4: Downloading progress bar

onCreate:
Coming from Scenarios 3a or 3b, searchActivity.java or listaSat2.java, the application implements two options.

3a: *(left screen)* method `doSearch` from searchActivity.java starts a progress bar while an `Asynctask` "`MiTarea`" is being performed in background. This is the longest process carried out in ZATDROID. With a quick internet connection, it lasts around 8 seconds, more than required. `MiTarea`:

- Retrieves key words typed by the user as a String
- Loads URL from every txt file in `celestrak.com`. `loadURLTLE` method matches these results.
- Reads all txt files with TLEs information from celestrak.com and write a unique xml file with all satellites together and their TLEs parameters. The xml format has been previously described. Also XSD Scheme is written. Both files are stored in the internal memory.
  TLEs are classified into files according to subtypes. So in this case, with a name based search all files must be read.
- Updates progress bar while the process is being carried out.
- Searches into the XML any sats matching the key words typed by the user. `searchListaSatsSAXHandler` is in charge of managing the search. It extends `defaultHandler` and uses SAX `ElementRoot` and Element to go through all nodes and store the sat names that matches the criteria. A `listArray` is created with all these sat names
- `MiTarea` finishes and the progress bar disappears. Go to Scenario 5.

3b (right screen) `listaSats.java` in its `onCreate` method retrieves tipoSatEspecifico from `SharedPreferences` and select the URL from the txt file in celestrak.com via method `seleccionarUrlTLE` according to tipoSatElegido. It also starts the progress bar. Then the `Asynctask` “`MiTarea`”is created:

- Reads txt file from celestrak.com and writes one xml file following format rules previously detailed and also the XSD Scheme. Both files are stored in the internal memory.
- Updates progress bar while the process is being carried out.
- Searches into the XML for all the sat names invoking `ListaSatsSAXHandler` which is in charge of managing the search. It extends `defaultHandler` and uses SAX `ElementRoot` and Element to go through all nodes and store all sat name. A `listArray` is created with all these sat names
- `MiTarea` finishes and the progress bar disappears. Go to Scenario 5.

This process lasts less than the other option because it reads only one txt file from celestrack.com, not all of them and fits with the requirements.

onClick:
This screens disappear automatically. It is not clickable.

7.2.7. SCENARIO 5: SATELLITE NAMES LIST

General Description: `ListView` showing satellite names according to previous selections.

Figure 27. Scenario 5: Satellite names list

onCreate:
Coming from `searchActivity.java` (left screen, scenario 3a) or `listaSats.java` (right screen, scenario 3b):

- `ListView` is created, `lista_sats.xml` layout complemented with code layout for *BreadCrumbs*
- *BreadCrumbs* at the top supporting English and Spanish. Here it can be seen the only one difference between screens. If coming from scenario 3a, the complete address is not shown, as there has not been any type-based selection. So *BreadCrumbs* only allows to navigate home.

onClick:

- *BreadCrumbs* `Buttons`: navigate backwards.
- `onListItemclick`: the name is saved in `SharedPreferences` as “satElegido”.
- The system searches into the XML for the sat picked. `satElegidoSAXHandler` which is in charge of managing the search. It extends `defaultHandler` and uses SAX `ElementRoot` and `Element` to go through all nodes and store the information only from the sat picked. A “sat” Java object is created with all these parameters. Sat is declared as `Parcelable`, so that it can be accessed from every class.
- Go to Scenario 6.

7.2.8. SCENARIO 6: FUNCTIONALITIES MAIN MENU

General Description: options menu

Figure 28. Scenario 6: Functionalities main menu

onCreate:

- Layout `menu_apps.xml`
- *BreadCrumbs* at the top supporting English and Spanish. (as shown in images) It retrieves “satElegido” and “tipoSatEspecifico” from `SharedPreferences` to show the complete route of selections.
- Icons represents the two functionalities. (English and Spanish version is displayed)) Buttons are created with XML resources setting states changing colours when clicked and modifying visual characteristics. The images are real screenshots.

onClick:

- Sat `parcelable` is retrieved.
- *BreadCrumbs* `buttons`: navigate backwards
- Top button: calls `mapsActivity.java` opening GOOGLE MAPS view. Go to Scenario 7.
- Bottom button: calls `ARIntro.java` starting the augmented reality view. While the view is loading, a dialog spinner-style pops up. (Figure 27a.). Go to Scenario 8.

Figure 28a. Scenario 6: loading spinner dialog

7.2.9. SCENARIO 7: Google Maps VIEW

General Description: GOOGLE MAPS View

Figure 29. Scenario 7: Google Maps View

onCreate:

- Layout "map" provided by GOOGLE MAPS. Some layers overlays the `MapView`
- It retrieves `parcelable` "sat" with TLE information of the picked satellite
- `CalculationsTime` object is created to modify time format
- `CalculationsSatPOS` is performed and sat Java object is completed with latitude, longitude and altitude of the satellite. These calculations will be detailed as their implementation in section 8
- `Geopoint` with latitude and longitude of the sat is created and map is centred onto this point
- List of overlays is defined:
  - o Compass is added to the view as a new layer
  - o `MapView` GOOGLE class is added as another layer. This class contains the GOOGLE MAPS information.

`MapView` class:

- ✓ A first trajectory line is drawn: 10 `GeoPoints` for the past 10 minutes and 10 `GeoPoints` for predicted 10 minutes are calculated thanks to `CalculationsSatPOS` which gives latitude , longitude and altitude, known time and sat object. These `GeoPoints` are joined with a line using Path object and there it is the trajectory.
- ✓ Then method "draw" is refreshing the view instantly with all these objects:
  - o `Geopoint` of the sat, together with an icon using `BitMap` and `Canvas`.
  - o Trajectory, recalculating and redrawing it. It is limited to a 1 second refresh because the calculations are too complex and the line could deform. This is why a little delay can be notice when moving zooming or translating the map
  - o `Canvas.drawText` shows text with name, latitude longitude and altitude of the satellite at the top left corner of the screen.

onClick:

this screen is not clickable. The back Button from ANDROID closes this view and comes back to Scenario 6

7.2.10. SCENARIO 8a: NEXT OVERHEAD PASS

General description: In case satellite elevation is less than 0 (under the horizon) a message with next overhead pass is shown.

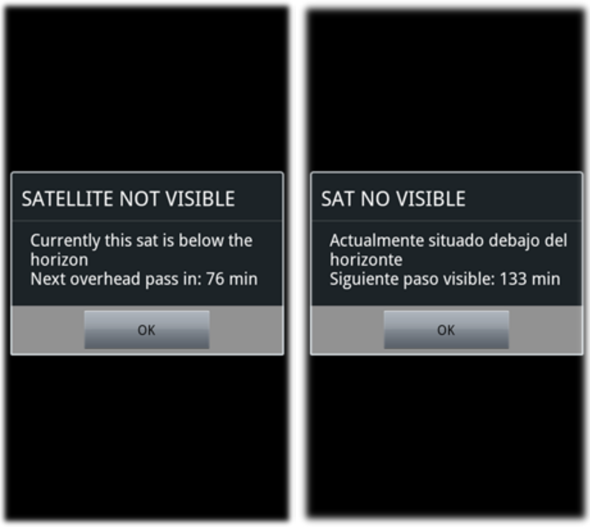


Figure 30. Scenario 8a: Next overhead pass

onCreate:

- `ARIntro.java` is called
- It retrieves `parcelable` "sat" with TLE information of the picked satellite
- `DevicePositionProvider` is called for latitude, longitude and altitude from device. It uses `LocationManager`, GPS, Network and web Service GOOGLE MAPS elevation if necessary.
- `CalculationsTime` modifies format date.
- Orbital Mechanics is developed by `CalculationsSatTRACK` method, where knowing sat object, device position and current time it calculates elevation and azimuth of the satellite from the place where the device is located.
- Just in case sat elevation is lower than 0, the satellite is under the horizon, and the message of figure 29 gives information about it. Moreover, calculations inside a loop are developed increasing the current date and time, until the method knows when it will be the next overhead pass. This is also added to the message.

onClick:

- Next scenario is reached automatically when ok button is pressed: `ARCameraActivityOverlay.java` is started.

7.2.11. SCENARIO 8b: AUGMENTD REALITY VIEW

General description: Augmented reality view positioning satellite in the sky.

Figure 31. Scenario 8b: augmented Reality view

onCreate:

`ARCameraActivityOverlay` is started and manages all the process to create the complete view.

- Implements `SensorEventListener`, so it is update every time a sensor changes.
- It retrieves `parcelable` “sat” with TLE information of the picked satellite
- The Camera `SurfaceView` is created (`ARCameraPreview_Overlay.java`) where the camera is started.
- Now layers are managed with OPENGL. Each layer has a `GLSurfaceView`, a render and an object.
- 3 `GLSurfacesViews` are also created as layer on top of the Camera view:
    - o `AROurSurfaceOverlaySat`: draw the icon and uses sat Render.
    - o `AROurSurfaceOverlayCruz`: draw central square and uses crus Render.
    - o `AROurSurfaceOverlayFlecha`: draw arrow indicating the location of the sat. Uses fleche render.
- These `GLSurfaceViews` uses `Renders` created also in `ARCameraActivityOverlay`.
    - o `ARGLRenderOverlayMovSat`:
        - It creates `ARGLSat` object which sets the geometrics of the Sat.
        - Receive gravity vector, azimuth and elevation of device and smooth the values to make the view more stable
        - GL10 object is created
        - `onDrawFrame` method, updated instantly, manages OPENGL camera with `GLU.gluLookAt` changing vectors between coordinate systems, draws the icon when orientation from device and sat matches or is near (percentage calculation) and configure screen OPENGL view limits.
    - o `ARGLRenderOverlayMovCruz`: similar to previous render, but with the central square. It changes colour when orientation from device and sat matches exactly. The sat icon is inside the square. It creates `ARGLCruz` object.
    - o `ARGLRenderOverlayMovFlecha`: similar with the arrow. It indicates the direction in which the sat can be found. It helps the user to rotate the device in the right way. It disappears when orientation from device and sat matches in a certain percentage. It creates `ARGLFlecha` object.

- `DevicePositionProvider` gives the position of the satellite, taking advantage of GPS, network, location manager and, if necessary, web service GOOGLE MAPS Elevation with Http classes of ANDROID.
- Sensors accelerometer, magnetic and gravity are created.
- 2 `TextViews` with information of sat orientation and device orientation are created and added on top of the view.
- `onSensorChanged` method keeps the values updated. This method is executed whenever a sensor changes.
    - `RotationMatrix` with information of the coordinate system axis of the device is retrieved and then used to calculate the elevation and azimuth of the device in real time.
    - `TextViews` of the device is updated.
    - Device elevation and azimuth are passed to the renders.

Onclick:
this screen is not clickable. The back `Button` from ANDROID closes this view and comes back to Scenario 6

8. ORBITAL MECHANICS

This project makes sense since it involves a lot of knowledge in orbital mechanics. This section aims to be an introduction of the complex calculations used, as well as some general concepts concerning satellites. Taking advantage of the introduction in the final year project report section 6, some more detailed information is given below.

8.1. COORD. SYST., NEWTON`S & KEPLER`S LAWS. KEPLERIAN ELEMENTS.

All this information has been already introduced in project report. Taking all that concepts in mind, and playing with the formulas. It an be demonstrated that once the keplerian elements are known for a orbit, position and velocity can be calculated given a current time and a known initial condition.

Figure 32 shows the process and formulas that are used in the process to calculate position and velocity vectors out of keplerian elements. Detailed process with equations can be found in [2]

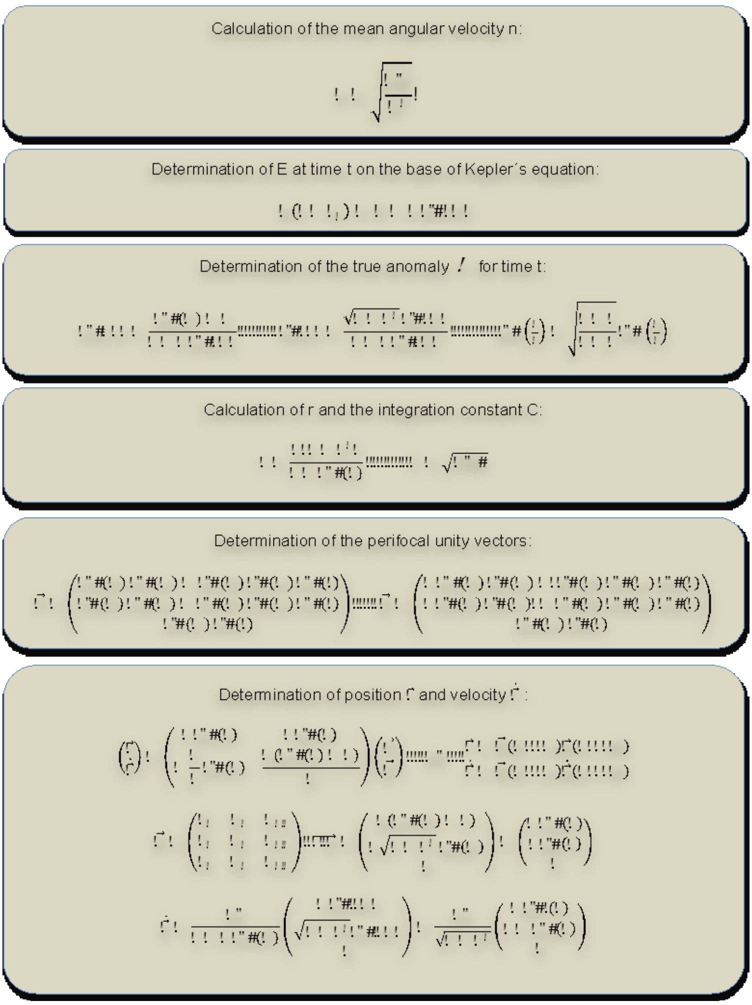


Figure 32. Diagram for orbit prediction with Newton´s equations.

Needless to say that this is a theoretical development that could be right in ideal conditions, based on the assumption that the motion of the satellite is a result of the gravitational attractions between two bodies. So it is clear that new and improved models need to be developed.

8.2. NORAD ORBIT PROPAGATION MODELS

SpaceTrack Report 3 [4] developed for the first time in 1980 the complete models to face orbits prediction and there is no better source to explain what they did:

"*NORAD maintains general perturbation element sets on all resident space objects. These element sets are periodically refined so as to maintain a reasonable prediction capability on all space objects. In turn, these element sets are provided to users. The purpose is to provide the user with a means of propagating these element sets in time to obtain position & velocity space object*"

"*The most important point to be noted is that not just any prediction model will suffice. The NORAD element sets are mean values obtained by removing periodic variations in a particular way. In order to obtain good predictions, these periodic variations must be reconstructed (by the prediction model) in exactly the same way they were removed by NORAD. Hence, inputting NORAD element sets into a different model (even though the model may be more accurate or even a numerical integrator) will result in degraded predictions. The NORAD element sets must be used with one of the models described in this report in order to retain maximum prediction accuracy*"

8.2.1. NORAD TLES

Two Line Element Sets TLEs are the files containing the element sets daily updated by NORAD. With this information and studying the models, the position and velocity of any object in space can be retrieved. Although format of TLEs has already been explained, figure 33 review it:

```
AAAAAAAAAAAAAAAAAAAAAAAA
1 NNNNNU NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNN +NNNNN-N +NNNNN-N N NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNN NNN.NNNN NNN.NNNN NN.NNNNNNNNNNNNNN
```

Line 1	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
08	Classification (U=Unclassified)
10-11	International Designator (Last two digits of launch year)
12-14	International Designator (Launch number of the year)
15-17	International Designator (Piece of the launch)
19-20	Epoch Year (Last two digits of year)
21-32	Epoch (Day of the year and fractional portion of the day)
34-43	First Time Derivative of the Mean Motion
45-52	Second Time Derivative of Mean Motion (decimal point assumed)
54-61	BSTAR drag term (decimal point assumed)
63	Ephemeris type
65-68	Element number
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)

Line 2	
Column	**Description**
01	Line Number of Element Data
03-07	Satellite Number
09-16	Inclination [Degrees]
18-25	Right Ascension of the Ascending Node [Degrees]
27-33	Eccentricity (decimal point assumed)
35-42	Argument of Perigee [Degrees]
44-51	Mean Anomaly [Degrees]
53-63	Mean Motion [Revs per day]
64-68	Revolution number at epoch [Revs]
69	Checksum (Modulo 10)

Figure 33. TLEs explanation

These data can be used together with SGP models to obtain accurate propagation of the orbit.

- First line contains identification data sets and perturbation influences by geopotential (The Earth is not an sphere) and the drag forces.
- The Second line provides with the keplerian elements expect for the semimajor axis, that can be determined easily.

These TLEs are provided at the moment by `Celestrak.com`, which was created and is maintained by Dr. T.S. Kelso, recognized worldwide as an expert in the area of satellite tracking and orbit determination. Regularly sought out for advice and counsel by NASA, European Space Agency (ESA), Russian Space Institute, US and AF Space Command, and many universities and commercial space organizations.

TLEs are txt files classified into several files by satellite types. There is a fixed URL for every satellite type, e.g., `http://celestrak.com/NORAD/elements/weather.txt` for weather satellites. ZATDROID implements lists that matches the type required with its URL and then a download module (UC-01) is executed.

8.2.2. NORAD PROPAGATIONS MODELS SGP4/SPD4

Five mathematical models for prediction of satellite position and velocity are available: [4]

- **SGP** (*Simplified General Propagation*) was developed by Hilton & Kuhlman (1966) and is used for near-Earth satellites. This model uses a simplification of the work of Kozai (1959) for its gravitational model and it takes the drag effect on mean motion as linear in time. This assumption dictates a quadratic variation of mean anomaly with time. The drag effect on eccentricity is modelled in such a way that perigee height remains constant.
- **SGP4** (*Simplified General Propagation version 4*) was developed by Ken Cranford in 1970 (see Lane and Hoots 1979) and is used for near-Earth satellites. This model was obtained by simplification of the more extensive analytical theory of Lane and Cranford (1969) which uses the solution of Brouwer (1959) for its gravitational model and a power density function for its atmospheric model (see Lane, et al. 1962).
- **SDP4** (*Simplified General Deep Space Perturbation model version 4)* is an extension of SGP4 to be used for deep-space satellites. The deep-space equations were developed by Hujsak (1979) and model the gravitational effects of the moon and sun as well as certain sectoral and tesseral Earth harmonics which are of particular importance for half-day and one-day period orbits."

In order to describe such disturbance forces, here it is a brief overview to them: [2]

SGP:

- **Perturbations due to Geopotential**
  The Earth is not spherical, in fact it has a bulge at the equator, is flattened at the poles and is slightly pear-shaped. This leads to perturbations in all Keplerian elements. The second order deformation of the Earth considers the fact that it is slightly flattened.

- **Perturbations due to the Sun and the Moon**
  The Sun and the Moon causes periodic variations in all Keplerian elements, but secular perturbations only to the right ascension of ascending node and the argument of perigee.
  These perturbations have their minima for the same inclinations, *i*, as the non spherical Earth perturbations and become larger for higher altitude orbits.

SGP4 / SPD4:

- **Perturbations due to Atmospheric Drag**
  The atmospheric drag is a force creating an acceleration, in the opposite direction of the satellites velocity. It is influenced by the density of the atmosphere, the drag coefficient, the cross section area and the mass of the satellite.
  The atmospheric drag is a breaking force and hence removes energy from the satellite in orbit. This leads to a decrease in orbital height, but at very low rates.

- **Perturbations due to Solar Radiation**
  The acceleration caused by solar radiation depends on a reflection factor between 1 and 0. The perturbations due to solar radiation is in the same magnitude as atmospheric drag perturbations for altitudes at 800 km, and less for lower orbits.

Detailed equations developed within the models can be found in *Appendix 1*, courtesy of [4].

8.3. IMPLEMENTATION

There are 11 classes created to implement these propagation models:

CLASS	INPUTS	OUTPUTS	DESCRIPTION
CalculationsSatPOS:	currentTime sat object	sat (with latitude, longitude and altitude)	Manage the calculation of latitude, longitude and altitude
CalculationsSatTRACK:	currentTime sat object device location	sat (with azimuth and elevation)	Manage the calculation of azimuth and elevation
CalculationsOrbit:	Time since epoch sat object	sat (with position and velocity)	Manages SGP4 or SDP4 modules
CalculationsOrbitSGP4:	sat object	sat (with position and velocity)	SGP4 model code
CalculationsOrbitSDP4:	sat object	sat (with position and velocity)	SDP4 model code
CalculationsVarSDP4:	Variables for SDP4		
CalculationsOrbitSDP4_Deep:	Variables and methods for SDP4		
CalculationsVarGlobal:	Variables for SDP4 and SGP4		
CalculationsTime:	Calculations with different time formats		
CalculationsMaths:	Maths functions		
k	constants		

Table 76. Classes for Orbital Mechanics Calculations

The sequence in which these classes are used is described in figure 34.

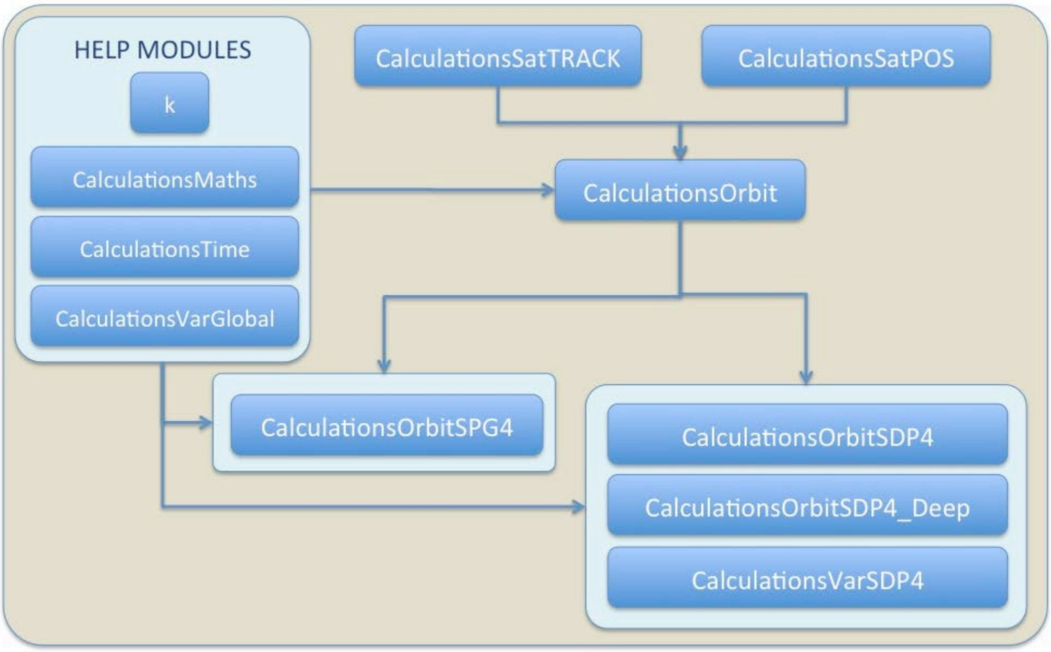


Figure 34. Orbital Mechanics Classes Diagram

- Help Modules contains variables, Maths functions, time functions.
- Both `CalculationsSatTRACK` and `CalculationSatPOS` use `CalculationsOrbit` to choose between SGP4 or SDP4 models.
- `CalculationsSatTRACK`: gets azimuth and elevation
- `CalculationSatPOS`: gets latitude, longitude and elevation
- `CalculationsOrbitsSDP4_Deep`: contains a complex module with equations and cases.

9. BREAKTHROUGH DESIGN FEATURES

9.1. MULTILANGUAGE SUPPORT

Following ANDROID developers advices, all strings used in ZATDROID have been encoded inside the XML file `strings.xml` in English, which is the main language. This file is located in *res/values* There is also a file `string-es.xml` in *res/values-es* in which all strings are duplicated Spanish.

Whenever the user runs ZATDROID, the language of the strings are shown in the language of the device automatically, as long as it is English or Spanish. English is by default.

Figure 35. Multilanguage Screen

9.2. MULTIPLE SCREENS SUPPORT: ICONS, TEXT

Bearing in mind the wide range of devices it is mandatory to support different screen sizes. Although in the ANDROID help it is recommended the use of "*dp*" (*density pixels*) when defining pictures or text size in XML layouts, it has not been the solution this time.

Many tests have been run with different devices (*Sony Xperia Neo V*, *Samsung Note*, *Samsung Galaxy S4*, *Samsung Tablet 11'*, *Nexus 4*, *Samsung S3Mini*) and using "*dp*" did not show the icons and text the same way in all devices. So another definition of the layouts was necessary.

Finally, it was found out that using only xml layouts was not the right procedure. Instead, layouts definition inside Java code was selected. This option lets the programmer customize deeper the layout configuration. `addView` adds views (layouts) to other layouts.

- On the one hand, icons needs to set height and width according to the size of the device. This is achieved by retrieving the device width and height with this piece of code:
```
final float height=getResources().getDisplayMetrics().heightPixels
final float width=getResources().getDisplayMetrics().widthPixels
```
  Every icon height and width can be configured as a certain percentage of total:
```
int h = (int) (0.05 * height); // 5%
int w = (int) (0.20 * width);  // 20%
```
- On the other hand, text size (`setTextSize`) and gap between letters (`setTextScaleX`) is something more complex. The text must adjust to the size of its box perfectly.

```
private float scaleButtonText (int h, int w, int p, String bText) {
  Paint paintRectText = new Paint();
  paintRectText.setTextSize(h); // Units are pixels here
  paintRectText.setTextScaleX(1.0f);
  Rect bounds = new Rect();
  // ask the paint for the bounding rect if it were to draw this text.
  paintRectText.getTextBounds(bText, 0, bText.length(), bounds);
  // determine the width
  int wText = bounds.right - bounds.left;
  // Calculate the new scale in x direction to fit text to the button width
  float TextScaleX = (float) (w - 2*p) / (float) wText;
  return TextScaleX;
}
```

Gap between letters:

A new function has been created `scaleButtonText`.

- A `Paint` object is created with 100% height and setTextScaleX = 1
- Ask the `paint` for the bounding rectangle if it were to draw this text.
- Determine the width
- Calculate the new scale in x direction to fit the text to the button width

Text Size:

It is set to a percentage of the total height of the box.

9.3. ASYNCTASK AND PROGRESS BAR: RUNNING CODE IN BACKGROUND.

There are two events within the application when a connection through internet is established to download files. This happens when connecting to CELESTRAK.COM and downloading TLEs in a txt file. Such a process may last some seconds and give errors. It is also the moment when both XML and XSD file is created from txt.

`Asynctask` is defined in the ANDROID developers site:

"This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called `Params`, `Progress` and `Result`, and 4 steps, called `onPreExecute`, `doInBackground`, `onProgressUpdate` and `onPostExecute`"

So this every time ZATDROID downloads a file from the internet, `asynctask` is launched and a progress bar is shown to the user to keep informed of the task that are being performed.

- Here it is an example of some of the code involving `Asynctask`:

```
public class MiTarea extends AsyncTask<String, Float, Integer>{
      protected ProgressDialog dialog;
      public MiTarea(ProgressDialog dialog) {
        this.dialog = dialog;
      }
      protected void onPreExecute() {
          dialog.setCancelable(true);dialog.setProgress(0);dialog.setMax(100);
          dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);dialog.show();
      }
      protected Integer doInBackground(String... sTipoGuardado) {
         // Load file txt from www.celestrack.com of the sat type chosen
        // Create xml & xsd files
        BufferedWriter xml = null, xsd = null;
        BufferedReader in = null;
        try {
         // Read file from the internet and save it in an internal file
         // Create a URL for the desired page
         URL url = new URL(sUrlTLE);
         publishProgress(10f);
         // Read all the text returned by the server
         in = new BufferedReader(new InputStreamReader(url.openStream()));
         publishProgress(30f);
         //Write xml file ...
         publishProgress(60f);
         //Write xsd file  ...
         publishProgress(65f);
      } catch (Exception e) {
      }
       return 100;
      }
```

```
        protected void onProgressUpdate (Float... valores) {
          int p = Math.round(valores[0]);
          dialog.setProgress(p);
        }
        protected void onPostExecute(Integer bytes) {
          dialog.dismiss();
        }
}
```

- To invoke this method :

```
ProgressDialog dialog = new ProgressDialog(this);
 Resources res = getResources();
 dialog.setMessage(res.getString(R.string.loading));
 new MiTarea(dialog).execute(sTipoGuardado);
```

9.4. SHARING AND STORING INFORMATION

ZATDROID requires to share information: files, java objects, parameters. There are many ways to share information in ANDROID.

SQLITE DATABASES:

In case a complex data management is needed with all the advantages of a relational data base. ZATDROID makes no use of this option because there is no need to manage a lot of information with tables and relationships. Also, XML files has been picked as the way to manage data because of its simplicity and the objective to learn concerning XML during this project.

PASSING OBJECTS: [9]

- SERIALIZABLE Class:

The problem with this approach is that reflection is used and it is a slow process. This mechanism also tends to create a lot of temporary objects and cause quite a bit of garbage collection.

- PARCELABLE Class

This code will run significantly faster. One of the reasons for this is that we are being explicit about the serialization process instead of using reflection to infer it. It also stands to reason that the code has been heavily optimized for this purpose.

To pass the PARCELABLE object between activities, `Intent.putExtra` is used. ZATDROID uses the “sat” class that implements PARCELABLE and passed between activities providing the information of the satellite picked.

- Methods implemented in the declaration:

```
public sat(Parcel in) {...

public void writeToParcel(Parcel dest, int flags) {{...

public static final Parcelable.Creator CREATOR = new Parcelable.Creator() {
  public sat createFromParcel(Parcel in) {
          return new sat(in);
}
```

- Save data:

```
startApp.putExtra("datosSatElegido", sat);
```

- Retrieve data:

```
Bundle data = getIntent().getExtras();
sat = data.getParcelable("datosSatElegido");
```

PASSING PARAMETERS:

- SHAREDPREFERENCES [10]

“The `sharedPreferences` class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use `sharedPreferences` to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).”

- Save data:

```
tipoGuardado = getSharedPreferences(fileName, 0);
SharedPreferences.Editor editor = tipoGuardado.edit();
editor.putInt("tipoSatGeneral", elegido);
editor.commit();
```

- Retrieve data:

```
tipoGuardado = getSharedPreferences(listaTipoSat.fileName, 0);
int iTipoGuardado = tipoGuardado.getInt("tipoSatGeneral",0);
```

ZATDROID uses `sharedPreferences` to share Strings that save names of satellites picked for the user.

- Final Static

  Constants are storaged in classes this way.

ZATDROID has a class named *k* for all constants used in mathematical calculations apart from other constants in other classes.

STORING FILES:

- External - Internal Storage: [11]

When building an app that uses the internal storage, the Android OS creates a unique folder, which will only be accessible from the app, so no other app, or even the user, can see what's in the folder.

The external storage is more like a public storage, so for now, it's the SD card, but could become any other type of storage (remote hard drive, or anything else).

The internal storage should only be used for application data, (preferences files and settings, sound or image media for the app to work). The external storage is often bigger. Besides, storing data on the internal storage may prevent the user to install other applications.

ZATDROID uses internal Storage to save the XML and XSD files because they are small and are used only for the application.

```
File archivo_xml = new File(getFilesDir() + File.separator + FILENAME_XML);
xml = new BufferedWriter(new FileWriter(archivo_xml));
```

9.5. BREADCRUMBS NAVIGATION BUTTONS

In every screen at the top the user can find some buttons as indicated in figure 8. These so-called `BreadCrumbs` functions are:

- To be helpful as they provide information of the satellite type picked during the process of several screens. All types are shown so that the user, at first sight, remembers the types and name of the satellite picked.
- To allow the user to navigate through the previous screens, changing the type picked, just by clicking on one of the buttons of the breadcrumbs.

The layouts affected are created programmatically, not with XML. This is another example where layouts defined with code allow to add some more functionalities than with XML.

Figure 36. BreadCrumbs

Examples can be seen in the code. They are too large to bring examples to this document.

9.6. GOOGLE SEARCH ACTIVITY

The first screen gives the opportunity to select searching a satellite by it type or launching a search by some key words. See Figure 9.

This last functionality uses GOOGLE Search Activity from Android. The button calls `onSearchRequested` method and the GOOGLE interface is launched. It opens a dialog with a keyboard where you can write key words to find a satellite. It also supports voice search.

It opens `searchActivity` and calls `onNewIntent` method, then `handleIntent`, where the string is retrieved and starts the process with `doSearch` method.

Figure 37. GOOGLE Search & progress bar

CELESTRAK provides txt files with TLEs classified by type. To be able to perform the search in all satellite files provided by CELESTRAK, ZATDROID needs to download all files, merge them together in one file and then search. This process lasts some seconds, it is encoded inside an `Asynctask` and a progress bar is shown to the user.

```
private void handleIntent(Intent intent) {
    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
      String query =
      intent.getStringExtra(SearchManager.QUERY);
      this.query = query;
      doSearch(query);
    }
  }
  private void doSearch(String queryStr) { ... }
}
```

9.7. DEVICE LOCATION PROVIDER

Latitude, longitude and altitude of the device must be known whenever a prediction for the location of the satellite in the sky is intended to do. Besides knowing the satellite georeference, also device coordinates are essential.
With the help of [12] ZATDROID implements a customized `LocationListener` to retrieve **latitude** and **longitude** coordinates:

- Checks that Network or GPS is enabled. If not, it shows a message to the user requiring at least one connection.
- Firstly retrieves location from Network provider. It is quicker and usually devices enables network but not always GPS.
- Secondly, if GPS is enabled, as it is more precise, uses it to get a more accurate location.

Finally, **altitude** is also required.

- If GPS is enabled, it is easy. GPS gives it together with latitude and longitude.
- Otherwise, a web service is called using a `httpRequest`. There are two possibilities based on 3D maps of the Earth: [13]
    - USGS Elevation Query Web Service: [14]
    - GOOGLE MAPS Elevation API Web Service: [15]

ZATDROID implements GOOGLE MAPS `Elevation API Web Service`, although there is no big difference between them, advantages or disadvantages.

- Retrieve latitude and longitude from GPS and/or Network:

```
public Location getLocation() {
  try {
   locationManager = (LocationManager) mContext.getSystemService(LOCATION_SERVICE);
   // getting GPS status
   isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
   // getting network status
   isNetworkEnabled = locationManager.isProviderEnabled(LocMngr.NETWORK_PROVIDER);
   if (!isGPSEnabled && !isNetworkEnabled) {
     // no network provider is enabled
   } else {
     this.canGetLocation = true;
     // First get location from Network Provider
     if (isNetworkEnabled) {
       locationManager.requestLocationUpdates(
       LocationManager.NETWORK_PROVIDER,MIN_TIME_BW_UPDATES,
       MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
       if (locationManager != null) {
         location = locationManager.getLastKnownLocation(LtnMngr.NETWORK_PROVIDER);
         if (location != null) {
           latitude = location.getLatitude();
           longitude = location.getLongitude();
         }
       }
     }
```

```
// if GPS Enabled get lat/long using GPS Services
     if (isGPSEnabled) {
       if (location == null) {
         locationManager.requestLocationUpdates(
         LocationManager.GPS_PROVIDER, MIN_TIME_BW_UPDATES,
         MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
         if (locationManager != null) {
           location = locationManager.getLastKnownLocation(LocatMar.GPS_PROVIDER);
           if (location != null) {
             latitude = location.getLatitude();
             longitude = location.getLongitude();
           }
         }
       }
     }
}
```

- Connecting to Web service GOOGLE MAPS elevation:

```
// Write the url for the request
HttpClient httpClient = new DefaultHttpClient();
HttpContext localContext = new BasicHttpContext();
String url = "http://maps.googleapis.com/maps/api/elevation/"
  + "xml?locations=" + String.valueOf(latitude)
  + "," + String.valueOf(longitude)
  + "&sensor=true";
HttpGet httpGet = new HttpGet(url);

// Process the answer
HttpResponse response = httpClient.execute(httpGet, localContext);
HttpEntity entity = response.getEntity();
```

9.8. SENSORS MANAGEMENT

Method `mSensorManager` provides access to all sensors installed in the device. Three sensors are to be used by ZATDROID:

- **Accelerometer**: measures the acceleration applied to the device that gives the orientation.
- **Magnetic Field**: magnetic vector is provided device coordinate system.
- **Gravity**: Gravity vector is given in device coordinate system.

Vectors need a reference coordinate system and as the device is moving and rotating, the transformation matrix is also necessary. Two methods inside `mSensorManager` (`getRotationMatrix` and `getOrientation`) give information about the orientation and coordinate system of the device and the transformation matrix. `onSensorChanged` is where all changes in the measurements are noticed and sent to the code.

- Declaring Sensors:

```
private SensorManager mSensorManager;
private Sensor mAccelerometer, mMagnetic, mGravity;
mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
mMagnetic = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
mGravity = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
mSensorManager.registerListener(this, mMagnetic,SensorManager.SENSOR_DELAY_UI);
mSensorManager.registerListener(this, mAccelerometer, 300000);
mSensorManager.registerListener(this, mGravity, SensorManager.SENSOR_DELAY_UI);
```

- Retrieveing measurements:

```
public void onSensorChanged(SensorEvent event) {
      // TODO Auto-generated method stub
      switch (event.sensor.getType()) {
      case Sensor.TYPE_ACCELEROMETER:
      System.arraycopy(event.values, 0, vGrav, 0, 3);
      break;
      case Sensor.TYPE_MAGNETIC_FIELD:
      System.arraycopy(event.values, 0, vMag, 0, 3);
      break;
      case Sensor.TYPE_GRAVITY:
      System.arraycopy(event.values, 0, vGravSensorGrav, 0, 3);
      break;
}
```

- Rotation Matrix and Orientation vector:

```
mSensorManager.getRotationMatrix(vR, vI, vGrav, vMag);
mSensorManager.getOrientation(vR, vOrient);
```

9.9. SMOOTH MOVEMENTS FILTER [16]

The augmented reality functionality requires to retrieve device orientation in real time. This information is used to locate the satellite icon on the screen when the user is rotating the device searching the right orientation (azimuth & elevation). Then, icon is moving through the screen.

This movement depends on the accuracy, updating time and speed of rotating the device. One method has been implemented to smooth the movement of the icon on the screen: `filterSmoothMovements`. It takes two factors into account:

- `SmoothFactorCompass`: so that the small jumps do not disturb
- `SmoothThresholdCompass`: minimum distance so that the icon jumps

```
private double filterSmoothMovements(double oldValue, double newValue) { //Degrees
/* The easing float that defines how smooth the movement will be
 *(1 is no smoothing and 0 is never updating, my default is 0.5).
  *We will call it SmoothFactorCompass.
 * The threshold in which the distance is big enough to turn immediately
 * (0 is jump always, 360 is never jumping, my default is 30).
 * We will call it SmoothThresholdCompass */
double SmoothFactorCompass = .4; //factor so that the small jumps do not disturb
double SmoothThresholdCompass = 30.0; // minimum distance so that the icon jumps

if (Math.abs(newValue - oldValue) < 180) {
  if (Math.abs(newValue - oldValue) > SmoothThresholdCompass) {
  temp = newValue;
  } else {
    temp = oldValue + SmoothFactorCompass * (newValue - oldValue);
  }
} else {
  if (360.0 - Math.abs(newValue - oldValue) > SmoothThresholdCompass) {
    temp = newValue;
  } else {
    if (oldValue > newValue) {
      temp = (oldValue + SmoothFactorCompass *
      ((360 + newValue - oldValue) % 360) + 360) % 360;
    } else {
      temp = (oldValue - SmoothFactorCompass *
      ((360 - newValue + oldValue) % 360) + 360) % 360;
    }
  }
}
return temp;
```

9.10. MIGRATION OF SGP4 CODE FROM FORTRAN / C TO JAVA FOR ANDROID

Document [4] stated the definition of the methods to calculate the propagation of the orbits taking into account perturbations due to disturbance forces. This was in 1980 in FORTRAN language. A new revision was developed in 2006 in C [6] Some other reviews and improvements have been implemented but the heart still remains.

ZATDROID implements the code from 1980 with some modifications (not all) from 2006 migrated to JAVA for ANDROID.

9.11. AUGMENTED REALITY VIEW: LAYERS OVER CAMERA VIEW.

Augmented Reality views means to be able to merge the camera view with some other artificial layers with digital information and let both worlds interchange information and interact with each other.

This has been achieved thanks to the `GLSurfaceView` and `SurfaceView` classes and then overlapping them with `addViews`. All these layouts have been created programmatically, as already explained before to increase customization.

ZATDROID implements 6 layers at the same time: (Figure 10)

1) The camera view, as usual: `ARCameraPreview_Overlay` (`SurfaceView`)
2) A rectangle in the centre that changes colour when device orientation is directly pointing at the satellite in the sky.
   a. `AROurSurfaceOverlayCruz` (`GLSurfaceView`)
   b. `ARGLRenderOverlayMovCruz` (`Renderer`)
   c. `ARGLCruz`: class object
3) An arrow indicating the direction of the satellite to rotate the device
   a. `AROurSurfaceOverlayFlecha` (`GLSurfaceView`)
   b. `ARGLRenderOverlayMovFlecha` (`Renderer`)
   c. `ARGLFlecha`: class object
4) An icon symbolizing the satellite in the right azimuth and elevation.
   a. `AROurSurfaceOverlaySat` (`GLSurfaceView`)
   b. `ARGLRenderOverlayMovSat` (`Renderer`)
   c. `ARGLSat`: class object
5) `TextView` with the name and orientation (azimuth and elevation) of the satellite
6) `TextView` with elevation and azimuth of the device updated in real time.

Figure 38. Augmented reality view

9.12. OPENGL USAGE

When implemented AR View, it was complicated to visualize the icon of the sat only when the device was pointing at the direction of the satellite. OPENGL has been used to rotate the virtual camera that shows the icon at the same time as the device camera is rotated by the user.

- `GLU.gluLookAt` and `GL10.glFrustumf` control the focus, position and movements of the virtual camera.
- `GL10` class manages all other OPENGL requirements.
- As explained before, the moving coordinate systems made the vectors transformations really complicated. Complex matrixes were used.
- The geometry of the arrow and rectangle was easy, but the icon was inside a rectangle and was always moving, so updating the geometry of the icon really increased the difficulty.

9.13. GOOGLE MAPS VIEW

As one of the two main functionalities, locating the satellite icon in the GOOGLE MAPS in real time with latitude and longitude is exciting. These maps have the same appearance as the maps you see in your computer, but here you can customize to show whatever you want to (Figure 10)

- `MapActivity` is the class extended in ZATDROID. Newer versions of ANDROID has deprecated `MapView` object and replaced it with `GoogleMap` object. The heart is similar but other complements are improved. For this app, `MapView` fits all requirements.
- `com.google.android.maps.Overlay` is the layer where all the information is added and shown onto the map.
- A canvas let the program add the satellite icon as a bitmap together with the `geoPoint` represented by the latitude and longitude of the satellite.
- Also in this canvas a `drawPath` object allows to draw a trajectory line. This trajectory represents the orbit in its last 20 minutes and its predicted 20 coming minutes, all calculated iterating the propagating models.
- And the last layer is the `textView` with name, latitude and longitude of the satellite updated in real time.

The satellite icon is updated almost instantly. The refreshing delay for the trajectory is one second though, that is, the trajectory stays old when zooming or moving the map. It was tested to update the trajectory several times per second so that the visual appearance was smoother, but the high amount of calculations, made it not possible for the program.

If the user ever sees a satellite at high altitudes (30.000 Km) and the trajectory is not drawn, it is not a bug. The satellite follows a geostationary orbit and therefore it is fixed. Actually, it is rotating with the Earth. For an inhabitant it is always in same position, same latitude and longitude.

Figure 39. GOOGLE MAPS View

9.14. XML/XSD MANAGEMENT

The TLEs downloaded from CELESTRAK is provided in txt format as described in appendix 1. One of the objectives of this project was to learn XML language. Therefore, the data management in ZATDROID was in XML. So TLEs are transformed into XML. The XML format is shown in appendix 2.

The way to create XML from the txt is done manually, by parsing txt NORAD format and then writing XML tags and content. The files are storaged in the internal memory, as explained before in this section.

```
protected void escribirXML(BufferedWriter xml, BufferedReader in) {
...
```

- Read txt from URL:

```
  while (continuar) {
    // //Read txt file and parse it //////////////
    linea1 = new StringBuilder(aux);
    // //// Erase spaces after the name
    while (linea1.toString().endsWith(" ")) {
    linea1.delete(linea1.toString().length() - 1, linea1.toString().length());
    }
    // & simbol give an error in the name. Erase it.
    if (linea1.toString().contains("&") ){
      linea1.deleteCharAt((linea1.toString().indexOf("&")));
    }
  // second line
  try {
    linea2 = in.readLine();
    // read the whole line
    linea2 = linea2.trim().replaceAll(" +", " "); // Leave only one space
    splitLinea2 = linea2.split(" "); // parse it
    // Third line
    linea3 = in.readLine(); // read the whole line
    linea3 = linea3.trim().replaceAll(" +", " "); // Leave only one space
    splitLinea3 = linea3.split(" "); // parse it
    } catch (IOException e1) {
      e1.printStackTrace();
  }
```

- Write XML:

```
xml.write(INI + sat);
xml.newLine();
xml.write(INI + info);
xml.newLine();
xml.write(INI + name + linea1.toString() + END + name);
xml.newLine();
xml.write(INI + number + splitLinea2[1] + END + number);
xml.newLine();
xml.write(END + info);
xml.newLine();
xml.write(INI + keplerianElements);
...
```

9.15. SAX MANAGEMENT

SAX (*Simple API for XML*) and DOM (*Document Object Model*) are the two methods to deal with XML data in ANDROID. A brief description is now given: [17]

SAX:

- Parses node by node
- Doesn't store the XML in memory
- We can´t insert or delete a node
- SAX is an event based parser
- SAX is a Simple API for XML
- Doesn't preserve comments
- SAX generally runs faster than DOM

DOM:

- Stores XML document into memory before processing
- Occupies more memory
- We can insert or delete nodes
- Traverse in any direction.
- DOM is a tree model parser
- Document Object Model (DOM) API
- Preserves comments

ZATDROID is:

- Dealing with big documents
- Not inserting nodes, just reading
- Intending not to use a lot of memory
- Not reading the whole document, just the node required.
- Not needing to store the whole document and creating the DOM tree.

So SAX was decided to be the method for the searches into XML. ZATDROID looks into the XML for the satellite picked by the user and read all the information of this certain satellite to create afterwards a "sat" JAVA object.

`Element` and `ElementRoot` are used to navigate through the nodes of the XML and then checking conditions and saving data. In this case, it goes through all nodes, search "name" Element and retrieve the value of all of the nodes in a list of arrays.

```
public class listaSatsSAXHandler extends DefaultHandler {
  private ArrayList<String> ListaNombres = new ArrayList<String>();
  public ArrayList<String> getData() {
    return ListaNombres;
  }
  public void parse(InputStream is) {
    String NAMESPACE = "http://www.w3schools.com";
    RootElement root = new RootElement(NAMESPACE, "TLE");
    // Just Name is read to show in the list
    Element E_sat = root.getChild(NAMESPACE, "sat");
    Element E_info = E_sat.getChild(NAMESPACE, "info");
    Element E_name = E_info.getChild(NAMESPACE, "name");
    E_name.setEndTextElementListener(new EndTextElementListener() {
      Public void end(String body) { // body is the label informations
        ListaNombres.add(body);
      }
    });

  try {
  Xml.parse(is, Xml.Encoding.UTF_8, root.getContentHandler());
  } catch (SAXException e) {
    Log.e("SAX XML", "sax xml.parse ", e);
  }
}
```

10. TESTING AND VALIDATING

- Mathematics functions have a method test which check them
- Time functions, preparing date format (*JD, NORAD, calendar*) for the NORAD models have also a test method inside `CalcualtionsTime.class` where all functions are checked.
- One extra app has been developed to implement tests:
- App Creation: activities management
    - Sensors Management
    - `RotationMatrix` and Orientation vector. This study of the coordinate system transformations, between device coordinate system and Earth Coordinate system has been really difficult and a lot of time has been invested on understanding the meaning of these values.
    - Augmented Reality view, overlaying camera view with `TextViews`, geometry.
    - OPENGL geometry creation and virtual camera management: 2D and 3D.
    - Device position provider examples; checking data with internet web sites retrieving latitude, longitude and altitude of the location.
    - Device orientation: checking with compass and experiment manually.
- And finally, orbital mechanics: results in terms of latitude, longitude, altitude, azimuth and elevation, as well as position and velocity of the satellites, are the final values to be checked.
    - GPREDICT [1] is a free software application that gives all these values. This software is considered to be reliable. As free software, source code in C language can be downloaded and it contains two tests (`test_001.c` and `test_002.c`). These files consist of satellite TLE data, a date and time and their SPG4/SPD4 orbit prediction position and velocity vectors. These same tests have been performed with ZatDroid and after some reviews, the code has been tested against GPREDICT successfully.
      `listaSats.class` contains test code against GPREDICT.
    - AIAA 2006-6753 paper [6] provides the code for SGP4/SDP4 in C, Pascal and FORTRAN. ZATDROID has also been tested against this paper.
    - Spacetrack Report #3 [4] also gives test modules that has been checked.
- Web sites tracking satellites can also provide these data: [18], [19]

11. LIST OF REFERENCES

[1] Csete, A. (2009). GPREDICT: Free, Real-Time Satellite Tracking and Orbit Prediction Softtware. Retrieved from `http://gpredict.oz9aec.net/`

[2] Daum, P. (2005). VIS SAT: A Satellite Footprint Visualization Tool (Master Thesis). Lancaster University.

[3] Grzegorczyk , M. (2013). SATFINDER. Retrieved from `http://esys.com.pl/satfinder`

***[4]** Hoots, Felix R., & Roehrich, R. L. (1980). Spacetrack Report #3: Models for Propagation of the NORAD Element Sets. Colorado Springs, CO: U.S. Air Force Aerospace Defense Command.*

[5] SATELLITE AR. (2011). ANALYTICAL GRAPHICS, INC. RETRIEVED FROM HTTP://SPACEDATA.AGI.COM/MOBILEAPPS/ABOUT.HTM

[6] Vallado, D. A., Crawford, P., Hujsak, R., & Kelso, T. S. (2006). *Revisiting spacetrack report #3*. In Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference, 2006 (Vol. 3, pp. 1984–2071). (*AIAA-2006-6753*)

[7] Videotutorials YOUTUBE, `stackoverflow.com`

[8] WIKIPEDIA.COM

[9] `http://www.developerphil.com/parcelable-vs-serializable/`

[10] `developer.android.com/`

[11] `http://stackoverflow.com/questions/5092591/what-are-the-differences-among-internal-storage-external-storage-sd-card-and-r`

[12] `http://www.androidhive.info/2012/07/android-gps-location-manager-tutorial/`

[13] `http://stackoverflow.com/questions/1995998/android-get-altitude-by-longitude-and-latitude`

[14] `http://gisdata.usgs.gov/xmlwebservices2/elevation_service.asmx`

[15] `https://developers.google.com/maps/documentation/elevation`

[16] `http://stackoverflow.com/questions/4699417/android-compass-orientation-on-unreliable-low-pass-filter`

[17] `http://stackoverflow.com/questions/12140851/sax-vs-dom-in-android`

[18] `http://www.n2yo.com/`

[19] `http://www.isstracker.com/`

12. LIST OF ABBREVIATIONS

SDK: Software Development Kit
AR: Augmented Reality
NORAD: North American Aerospace Defence Command
TLEs: Two Line Elements Sets
XML: eXtensible Markup Language
XSD: XML Scheme Definition
SAX: Simple API for XML
SGP: Simplified General Propagation Model
SGP4: Simplified General Propagation Model Version 4
SDP4: Simplified General Deep Space Perturbation model version 4
GUI: Graphical User Interface
DOM Document Object Model
API: Application Programming Interface

13. APPENDICES

APPENDIX 1: SGP4: (*Simplified General Perturbation Model version 4*) [4]

$$a_1 = \left(\frac{k_e}{n_o}\right)^{\frac{2}{3}}$$

$$\delta_1 = \frac{3}{2}\frac{k_2}{{a_1}^2}\frac{(3\cos^2 i_o - 1)}{(1 - {e_o}^2)^{\frac{3}{2}}}$$

$$a_o = a_1\left(1 - \frac{1}{3}\delta_1 - {\delta_1}^2 - \frac{134}{81}{\delta_1}^3\right)$$

$$\delta_o = \frac{3}{2}\frac{k_2}{{a_o}^2}\frac{(3\cos^2 i_o - 1)}{(1 - {e_o}^2)^{\frac{3}{2}}}$$

$$n_o'' = \frac{n_o}{1 + \delta_o}$$

$$a_o'' = \frac{a_o}{1 - \delta_o}.$$

For perigee between 98 kilometers and 156 kilometers, the value of the constant s used in SGP4 is changed to

$$s^* = a_o''(1 - e_o) - s + a_E$$

For perigee below 98 kilometers, the value of s is changed to

$$s^* = 20/\text{XKMPER} + a_E.$$

If the value of s is changed, then the value of $(q_o - s)^4$ must be replaced by

$$(q_o - s^*)^4 = \left[[(q_o - s)^4]^{\frac{1}{4}} + s - s^*\right]^4.$$

Then calculate the constants (using the appropriate values of s and $(q_o - s)^4$)

$$\theta = \cos i_o$$

$$\xi = \frac{1}{a_o'' - s}$$

$$\beta_o = (1 - {e_o}^2)^{\frac{1}{2}}$$

$$\eta = a_o'' e_o \xi$$

$$\begin{aligned} C_2 = & \ (q_o - s)^4 \xi^4 n_o'' (1 - \eta^2)^{-\frac{7}{2}} \left[a_o'' \left(1 + \frac{3}{2}\eta^2 + 4 e_o \eta + e_o \eta^3 \right) \right. \\ & \left. + \frac{3}{2} \frac{k_2 \xi}{(1 - \eta^2)} \left(-\frac{1}{2} + \frac{3}{2}\theta^2 \right) (8 + 24\eta^2 + 3\eta^4) \right] \end{aligned}$$

$$C_1 = B^* C_2$$

$$C_3 = \frac{(q_o - s)^4 \xi^5 A_{3,0} n_o'' a_E \sin i_o}{k_2 e_o}$$

$$\begin{aligned} C_4 = & \ 2 n_o'' (q_o - s)^4 \xi^4 a_o'' {\beta_o}^2 (1 - \eta^2)^{-\frac{7}{2}} \left(\left[2\eta(1 + e_o \eta) + \frac{1}{2} e_o + \frac{1}{2}\eta^3 \right] - \frac{2 k_2 \xi}{a_o''(1 - \eta^2)} \times \right. \\ & \left. \left[3(1 - 3\theta^2) \left(1 + \frac{3}{2}\eta^2 - 2 e_o \eta - \frac{1}{2} e_o \eta^3 \right) + \frac{3}{4}(1 - \theta^2)(2\eta^2 - e_o \eta - e_o \eta^3) \cos 2\omega_o \right] \right) \end{aligned}$$

$$C_5 = 2(q_o - s)^4 \xi^4 a_o'' {\beta_o}^2 (1 - \eta^2)^{-\frac{7}{2}} \left[1 + \frac{11}{4}\eta(\eta + e_o) + e_o \eta^3 \right]$$

$$D_2 = 4 a_o'' \xi {C_1}^2$$

$$D_3 = \frac{4}{3} a_o'' \xi^2 (17 a_o'' + s) {C_1}^3$$

$$D_4 = \frac{2}{3} a_o'' \xi^3 (221 a_o'' + 31 s) {C_1}^4.$$

The secular effects of atmospheric drag and gravitation are included through the equations

$$M_{DF} = M_o + \left[1 + \frac{3 k_2 (-1 + 3\theta^2)}{2 a_o''^2 {\beta_o}^3} + \frac{3 {k_2}^2 (13 - 78\theta^2 + 137\theta^4)}{16 a_o''^4 {\beta_o}^7} \right] n_o'' (t - t_o)$$

$$\begin{aligned} \omega_{DF} = & \ \omega_o + \left[-\frac{3 k_2 (1 - 5\theta^2)}{2 a_o''^2 {\beta_o}^4} + \frac{3 {k_2}^2 (7 - 114\theta^2 + 395\theta^4)}{16 a_o''^4 {\beta_o}^8} \right. \\ & \left. + \frac{5 k_4 (3 - 36\theta^2 + 49\theta^4)}{4 a_o''^4 {\beta_o}^8} \right] n_o'' (t - t_o) \end{aligned}$$

$$- _{DF} = - _o + \left[-\frac{3k_2\theta}{a_o''^2 {\beta_o}^4} + \frac{3{k_2}^2(4\theta - 19\theta^3)}{2a_o''^4 {\beta_o}^8} + \frac{5k_4\theta(3 - 7\theta^2)}{2a_o''^4 {\beta_o}^8} \right] n_o''(t - t_o)$$

$$\delta\omega = B^* C_3 (\cos\omega_o)(t - t_o)$$

$$\delta M = -\frac{2}{3}(q_o - s)^4 B^* \xi^4 \frac{a_E}{e_o \eta}[(1 + \eta\cos M_{DF})^3 - (1 + \eta\cos M_o)^3]$$

$$M_p = M_{DF} + \delta\omega + \delta M$$

$$\omega = \omega_{DF} - \delta\omega - \delta M$$

$$- \;= - _{DF} - \frac{21}{2}\frac{n_o'' k_2 \theta}{a_o''^2 {\beta_o}^2} C_1 (t - t_o)^2$$

$$e = e_o - B^* C_4 (t - t_o) - B^* C_5 (\sin M_p - \sin M_o)$$

$$a = a_o''[1 - C_1(t - t_o) - D_2(t - t_o)^2 - D_3(t - t_o)^3 - D_4(t - t_o)^4]^2$$

$$\begin{aligned} \mathbb{L} = \; & M_p + \omega + - \; + n_o'' \left[\frac{3}{2} C_1 (t - t_o)^2 + (D_2 + 2{C_1}^2)(t - t_o)^3 \right. \\ & + \frac{1}{4}(3D_3 + 12C_1D_2 + 10{C_1}^3)(t - t_o)^4 \\ & \left. + \frac{1}{5}(3D_4 + 12C_1D_3 + 6{D_2}^2 + 30{C_1}^2D_2 + 15{C_1}^4)(t - t_o)^5 \right] \end{aligned}$$

$$\beta = \sqrt{(1 - e^2)}$$

$$n = k_e \Big/ a^{\frac{3}{2}}$$

where $(t - t_o)$ is time since epoch. It should be noted that when epoch perigee height is less than 220 kilometers, the equations for a and $\mathbb{L}$ are truncated after the C_1 term, and the terms involving C_5, $\delta\omega$, and δM are dropped.

Add the long-period periodic terms

$$a_{xN} = e\cos\omega$$

$$\mathbb{L}_L = \frac{A_{3,0} \sin i_o}{8k_2 a\beta^2}(e\cos\omega)\left(\frac{3+5\theta}{1+\theta}\right)$$

$$a_{yNL} = \frac{A_{3,0} \sin i_o}{4k_2 a\beta^2}$$

$$\mathbb{L}_T = \mathbb{L} + \mathbb{L}_L$$

$$a_{yN} = e\sin\omega + a_{yNL}.$$

Solve Kepler's equation for $(E+\omega)$ by defining

$$U = \mathbb{L}_T - \text{-}$$

and using the iteration equation

$$(E+\omega)_{i+1} = (E+\omega)_i + \Delta(E+\omega)_i$$

with

$$\Delta(E+\omega)_i = \frac{U - a_{yN}\cos(E+\omega)_i + a_{xN}\sin(E+\omega)_i - (E+\omega)_i}{-a_{yN}\sin(E+\omega)_i - a_{xN}\cos(E+\omega)_i + 1}$$

and

$$(E+\omega)_1 = U.$$

The following equations are used to calculate preliminary quantities needed for short-period periodics.

$$e\cos E = a_{xN}\cos(E+\omega) + a_{yN}\sin(E+\omega)$$

$$e\sin E = a_{xN}\sin(E+\omega) - a_{yN}\cos(E+\omega)$$

$$e_L = ({a_{xN}}^2 + {a_{yN}}^2)^{\frac{1}{2}}$$

$$p_L = a(1 - {e_L}^2)$$

$$r = a(1 - e\cos E)$$

$$\dot{r} = k_e \frac{\sqrt{a}}{r} e \sin E$$

$$r\dot{f} = k_e \frac{\sqrt{p_L}}{r}$$

$$\cos u = \frac{a}{r} \left[\cos(E + \omega) - a_{xN} + \frac{a_{yN}(e \sin E)}{1 + \sqrt{1 - {e_L}^2}} \right]$$

$$\sin u = \frac{a}{r} \left[\sin(E + \omega) - a_{yN} - \frac{a_{xN}(e \sin E)}{1 + \sqrt{1 - {e_L}^2}} \right]$$

$$u = \tan^{-1} \left(\frac{\sin u}{\cos u} \right)$$

$$\Delta r = \frac{k_2}{2p_L}(1 - \theta^2) \cos 2u$$

$$\Delta u = -\frac{k_2}{4{p_L}^2}(7\theta^2 - 1) \sin 2u$$

$$\Delta\text{-} \ = \frac{3k_2\theta}{2{p_L}^2} \sin 2u$$

$$\Delta i = \frac{3k_2\theta}{2{p_L}^2} \sin i_o \cos 2u$$

$$\Delta \dot{r} = -\frac{k_2 n}{p_L}(1 - \theta^2) \sin 2u$$

$$\Delta r\dot{f} = \frac{k_2 n}{p_L} \left[(1 - \theta^2) \cos 2u - \frac{3}{2}(1 - 3\theta^2) \right]$$

The short-period periodics are added to give the osculating quantities

$$r_k = r \left[1 - \frac{3}{2} k_2 \frac{\sqrt{1 - {e_L}^2}}{{p_L}^2} (3\theta^2 - 1) \right] + \Delta r$$

$$u_k = u + \Delta u$$

$$- _k = - + \Delta -$$

$$i_k = i_o + \Delta i$$

$$\dot{r}_k = \dot{r} + \Delta \dot{r}$$

$$\dot{rf}_k = \dot{rf} + \Delta \dot{rf}.$$

Then unit orientation vectors are calculated by

$$\mathbf{U} = \mathbf{M} \sin u_k + \mathbf{N} \cos u_k$$

$$\mathbf{V} = \mathbf{M} \cos u_k - \mathbf{N} \sin u_k$$

where

$$\mathbf{M} = \left\{ \begin{array}{l} M_x = -\sin - _k \cos i_k \\ M_y = \cos - _k \cos i_k \\ M_z = \sin i_k \end{array} \right\}$$

$$\mathbf{N} = \left\{ \begin{array}{l} N_x = \cos - _k \\ N_y = \sin - _k \\ N_z = 0 \end{array} \right\}.$$

Then position and velocity are given by

$$\mathbf{r} = r_k \mathbf{U}$$

and

$$\dot{\mathbf{r}} = \dot{r}_k \mathbf{U} + (\dot{rf})_k \mathbf{V}.$$

SDP4: (Simplified General Deep Space Perturbation model version 4) [4]

$$a_1 = \left(\frac{k_e}{n_o}\right)^{\frac{2}{3}}$$

$$\delta_1 = \frac{3}{2}\frac{k_2}{{a_1}^2}\frac{(3\cos^2 i_o - 1)}{(1-{e_o}^2)^{\frac{3}{2}}}$$

$$a_o = a_1\left(1 - \frac{1}{3}\delta_1 - {\delta_1}^2 - \frac{134}{81}{\delta_1}^3\right)$$

$$\delta_o = \frac{3}{2}\frac{k_2}{{a_o}^2}\frac{(3\cos^2 i_o - 1)}{(1-{e_o}^2)^{\frac{3}{2}}}$$

$$n_o'' = \frac{n_o}{1+\delta_o}$$

$$a_o'' = \frac{a_o}{1-\delta_o}.$$

For perigee between 98 kilometers and 156 kilometers, the value of the constant s used in SDP4 is changed to

$$s^* = a_o''(1-e_o) - s + a_E.$$

For perigee below 98 kilometers, the value of s is changed to

$$s^* = 20/\mathrm{XKMPER} + a_E.$$

If the value of s is changed, then the value of $(q_o - s)^4$ must be replaced by

$$(q_o - s^*)^4 = \left[[(q_o - s)^4]^{\frac{1}{4}} + s - s^*\right]^4.$$

Then calculate the constants (using the appropriate values of s and $(q_o - s)^4$)

$$\theta = \cos i_o$$

$$\xi = \frac{1}{a''_o - s}$$

$$\beta_o = (1 - {e_o}^2)^{\frac{1}{2}}$$

$$\eta = a''_o e_o \xi$$

$$\begin{aligned} C_2 = \; & (q_o - s)^4 \xi^4 n''_o (1 - \eta^2)^{-\frac{7}{2}} \left[a''_o (1 + \frac{3}{2}\eta^2 + 4e_o\eta + e_o\eta^3) \right. \\ & \left. + \frac{3}{2} \frac{k_2 \xi}{(1 - \eta^2)} \left(-\frac{1}{2} + \frac{3}{2}\theta^2 \right) (8 + 24\eta^2 + 3\eta^4) \right] \end{aligned}$$

$$C_1 = B^* C_2$$

$$\begin{aligned} C_4 = \; & 2n''_o (q_o - s)^4 \xi^4 a''_o {\beta_o}^2 (1 - \eta^2)^{-\frac{7}{2}} \left(\left[2\eta(1 + e_o\eta) + \frac{1}{2}e_o + \frac{1}{2}\eta^3 \right] - \frac{2k_2\xi}{a''_o(1 - \eta^2)} \times \right. \\ & \left. \left[3(1 - 3\theta^2) \left(1 + \frac{3}{2}\eta^2 - 2e_o\eta - \frac{1}{2}e_o\eta^3 \right) + \frac{3}{4}(1 - \theta^2)(2\eta^2 - e_o\eta - e_o\eta^3) \cos 2\omega_o \right] \right) \end{aligned}$$

$$\dot{M} = \left[1 + \frac{3k_2(-1 + 3\theta^2)}{2a''^2_o {\beta_o}^3} + \frac{3{k_2}^2(13 - 78\theta^2 + 137\theta^4)}{16a''^4_o {\beta_o}^7} \right] n''_o$$

$$\dot{\omega} = \left[-\frac{3k_2(1 - 5\theta^2)}{2a''^2_o {\beta_o}^4} + \frac{3{k_2}^2(7 - 114\theta^2 + 395\theta^4)}{16a''^4_o {\beta_o}^8} + \frac{5k_4(3 - 36\theta^2 + 49\theta^4)}{4a''^4_o {\beta_o}^8} \right] n''_o$$

$$\dot{-}_1 = -\frac{3k_2\theta}{a''^2_o {\beta_o}^4} n''_o$$

$$\dot{-} = \dot{-}_1 + \left[\frac{3{k_2}^2(4\theta - 19\theta^3)}{2a''^4_o {\beta_o}^8} + \frac{5k_4\theta(3 - 7\theta^2)}{2a''^4_o {\beta_o}^8} \right] n''_o.$$

At this point SDP4 calls the initialization section of DEEP which calculates all initialized quantities needed for the deep-space perturbations (see Section Ten).

The secular effects of gravity are included by

$$M_{DF} = M_o + \dot{M}(t - t_o)$$

$$\omega_{DF} = \omega_o + \dot{\omega}(t - t_o)$$

$$- _{DF} = - _o + \dot{-} (t - t_o)$$

where $(t - t_o)$ is time since epoch. The secular effect of drag on longitude of ascending node is included by

$$- = - _{DF} - \frac{21}{2} \frac{n_o'' k_2 \theta}{a_o''^2 \beta_o{}^2} C_1 (t - t_o)^2.$$

Next, SDP4 calls the secular section of DEEP which adds the deep-space secular effects and long-period resonance effects to the six classical orbital elements (see Section Ten).

The secular effects of drag are included in the remaining elements by

$$a = a_{DS} [1 - C_1 (t - t_o)]^2$$

$$e = e_{DS} - B^* C_4 (t - t_o)$$

$$I\!L = M_{DS} + \omega_{DS} + - _{DS} + n_o'' \left[\frac{3}{2} C_1 (t - t_o)^2 \right]$$

where a_{DS}, e_{DS}, M_{DS}, ω_{DS}, and $- _{DS}$, are the values of n_o, e_o, M_{DF}, ω_{DF}, and - after deep-space secular and resonance perturbations have been applied.

Here SDP4 calls the periodics section of DEEP which adds the deep-space lunar and solar periodics to the orbital elements (see Section Ten). From this point on, it will be assumed that n, e, I, ω, - , and M are the mean motion, eccentricity, inclination, argument of perigee, longitude of ascending node, and mean anomaly after lunar-solar periodics have been added.

Add the long-period periodic terms

$$a_{xN} = e \cos \omega$$

$$\beta = \sqrt{(1 - e^2)}$$

$$I\!L_L = \frac{A_{3,0} \sin i_o}{8 k_2 a \beta^2} (e \cos \omega) \left(\frac{3 + 5\theta}{1 + \theta} \right)$$

$$a_{yNL} = \frac{A_{3,0} \sin i_o}{4 k_2 a \beta^2}$$

$$I\!L_T = I\!L + I\!L_L$$

$$a_{yN} = e \sin \omega + a_{yNL}.$$

Solve Kepler's equation for $(E + \omega)$ by defining

$$U = I\!L_T - -$$

and using the iteration equation

$$(E + \omega)_{i+1} = (E + \omega)_i + \Delta(E + \omega)_i$$

with

$$\Delta(E + \omega)_i = \frac{U - a_{yN} \cos(E + \omega)_i + a_{xN} \sin(E + \omega)_i - (E + \omega)_i}{-a_{yN} \sin(E + \omega)_i - a_{xN} \cos(E + \omega)_i + 1}$$

and

$$(E + \omega)_1 = U.$$

The following equations are used to calculate preliminary quantities needed for short-period periodics.

$$e \cos E = a_{xN} \cos(E + \omega) + a_{yN} \sin(E + \omega)$$

$$e \sin E = a_{xN} \sin(E + \omega) - a_{yN} \cos(E + \omega)$$

$$e_L = ({a_{xN}}^2 + {a_{yN}}^2)^{\frac{1}{2}}$$

$$p_L = a(1 - {e_L}^2)$$

$$r = a(1 - e \cos E)$$

$$\dot{r} = k_e \frac{\sqrt{a}}{r} e \sin E$$

$$r\dot{f} = k_e \frac{\sqrt{p_L}}{r}$$

$$\cos u = \frac{a}{r} \left[\cos(E + \omega) - a_{xN} + \frac{a_{yN}(e \sin E)}{1 + \sqrt{1 - {e_L}^2}} \right]$$

$$\sin u = \frac{a}{r}\left[\sin(E+\omega) - a_{yN} - \frac{a_{xN}(e\sin E)}{1+\sqrt{1-{e_L}^2}}\right]$$

$$u = \tan^{-1}\left(\frac{\sin u}{\cos u}\right)$$

$$\Delta r = \frac{k_2}{2p_L}(1-\theta^2)\cos 2u$$

$$\Delta u = -\frac{k_2}{4{p_L}^2}(7\theta^2-1)\sin 2u$$

$$\Delta\text{-} \ = \frac{3k_2\theta}{2{p_L}^2}\sin 2u$$

$$\Delta i = \frac{3k_2\theta}{2{p_L}^2}\sin i_o \cos 2u$$

$$\Delta\dot{r} = -\frac{k_2 n}{p_L}(1-\theta^2)\sin 2u$$

$$\Delta r\dot{f} = \frac{k_2 n}{p_L}\left[(1-\theta^2)\cos 2u - \frac{3}{2}(1-3\theta^2)\right]$$

The short-period periodics are added to give the osculating quantities

$$r_k = r\left[1 - \frac{3}{2}k_2\frac{\sqrt{1-{e_L}^2}}{{p_L}^2}(3\theta^2-1)\right] + \Delta r$$

$$u_k = u + \Delta u$$

$$\text{-}_k = \text{-} \ + \Delta\text{-}$$

$$i_k = I + \Delta i$$

$$\dot{r}_k = \dot{r} + \Delta\dot{r}$$

$$r\dot{f}_k = r\dot{f} + \Delta r\dot{f}.$$

Then unit orientation vectors are calculated by

$$\mathbf{U} = \mathbf{M}\sin u_k + \mathbf{N}\cos u_k$$

$$\mathbf{V} = \mathbf{M}\cos u_k - \mathbf{N}\sin u_k$$

where

$$\mathbf{M} = \left\{ \begin{array}{l} M_x = -\sin\text{-}\ _k \cos i_k \\ M_y = \cos\text{-}\ _k \cos i_k \\ M_z = \sin i_k \end{array} \right\}$$

$$\mathbf{N} = \left\{ \begin{array}{l} N_x = \cos\text{-}\ _k \\ N_y = \sin\text{-}\ _k \\ N_z = 0 \end{array} \right\}.$$

Then position and velocity are given by

$$\mathbf{r} = r_k \mathbf{U}$$

and

$$\dot{\mathbf{r}} = \dot{r}_k \mathbf{U} + (r\dot{f})_k \mathbf{V}.$$

CONSTANTS DEFINITION

Variable name	Definition	Value
CK2	$\frac{1}{2}J_2{a_E}^2$	5.413080E-4
CK4	$-\frac{3}{8}J_4{a_E}^4$	.62098875E-6
E6A	10^{-6}	1.0 E-6
QOMS2T	$(q_o - s)^4\ (\mathrm{er})^4$	1.88027916E-9
S	$s\ (\mathrm{er})$	1.01222928
TOTHRD	$2/3$	.66666667
XJ3	J_3	-.253881E-5
XKE	$k_e \left(\frac{\mathrm{er}}{\mathrm{min}}\right)^{\frac{3}{2}}$	.743669161E-1
XKMPER	kilometers/Earth radii	6378.135
XMNPDA	time units/day	1440.0
AE	distance units/Earth radii	1.0
DE2RA	radians/degree	.174532925E-1
PI	π	3.14159265
PIO2	$\pi/2$	1.57079633
TWOPI	2π	6.2831853
X3PIO2	$3\pi/2$	4.71238898

ABBREVIATIONS

n_o = the SGP type "mean" mean motion at epoch

e_o = the "mean" eccentricity at epoch

i_o = the "mean" inclination at epoch

M_o = the "mean" mean anomaly at epoch

ω_o = the "mean" argument of perigee at epoch

- $_o$ = the "mean" longitude of ascending node at epoch

$\dot{n}_o$ = the time rate of change of "mean" mean motion at epoch

$\ddot{n}_o$ = the second time rate of change of "mean" mean motion at epoch

B^* = the SGP4 type drag coefficient

$k_e = \sqrt{GM}$ where G is Newton's universal gravitational constant and M is the mass of the Earth

a_E = the equatorial radius of the Earth

J_2 = the second gravitational zonal harmonic of the Earth

J_3 = the third gravitational zonal harmonic of the Earth

J_4 = the fourth gravitational zonal harmonic of the Earth

$(t - t_o)$ = time since epoch

USER

MANUAL

ZatDroid

Satellite Tracking and Augmented Reality App for Android

1. TABLE OF CONTENTS

1. Table of Contents 3
2. List of Figures 5
3. Introduction 7
4. What can I do with ZATDROID? Main functionalities 7
5. Minimum Device Requirements 8
6. Installation Instructions 8
7. Executing The App: Screen Sequence Diagram 9
7.1. Intro 9
7.2. Initial Search Menu 10
7.3. Satellite Search Proccess 11
7.4. Downloading Files From The Internet 12
7.5. Satellite Names List 13
7.6. Functionalities Main Menu 14
7.7. GOOGLE MAPS Funcionality 15
7.8. Augmented Reality Functionality 16
8. Troubleshooting 17

2. LIST OF FIGURES

Figure 1. Main functionalities. 7
Figure 2. Screen Sequence Diagram 9
Figure 3. Screen 1: Intro 9
Figure 4. Screen 2a: Initial Search Menu 10
Figure 5: Screen 2b: Internet Connection Checking 10
Figure 6: Screen 3a: Search by key words 11
Figure 7: Screen 3b: Types of satellites 11
Figure 8: Screen 4: Downloading TLEs Progress Bar 12
Figure 9: Screen 5: Listing satellite names from XML file 13
Figure 10: Screen 6: Functionalities Menu 14
Figure 11: Screen 7: GOOGLE MAPS Funcionality 15
Figure 12: Screen 8a: Visibility Message 16
Figure 13: Screen 8b: Augmented Reality View 16

3. INTRODUCTION

This document is concerned to explain the features that ZATDROID offers the users, the installation process and a brief description of the main functionalities. The objective of this application is to introduce all the satellites to the users, showing information about any of them; from the orbit parameters up to the characteristics of the satellite.

ZATDROID joins two worlds: space engineering and computer science. From the beginning, this project grew up with the idea of working in the two topics that the author has been studying recently. On the one hand, ZATDROID is an ANDROID App. with the new features applied to devices such as tablets or smartphones. On the other hand, orbital mechanics and the orbit predictions tools and equations are to be included in the project.

There has been a lot of training in the planning because of the new features included in the program with JAVA for ANDROID and also because of the complex calculations involved in the propagations models for orbits predictions. These models are being used at the moment by NASA or satellites companies to performed actual orbits predictions.

ZATDROID takes advantage of GOOGLE MAPS to show real time satellites locations, OPENGL render options, views layers to show Augmented Reality view.

4. WHAT CAN I DO WITH ZATDROID? MAIN FUNCTIONALITIES

Figure 1. Main functionalities.

Once a user has a smartphone or tablet with ANDROID and installed the *apk* file ZATDROID can run.

Basically, with ZATDROID the user:

- Picks a satellite:
    - From a specific type navigating through menus with icons
    - From a search with key words.
- Locates it in GOOGLE MAPS together with its trajectory updated in real time.
- Sees it in the sky through Augmented reality in camera view.

ZATDROID gives the users the opportunity to interact with the satellites from their own devices in short time and simply with some clicks on your device. All those satellites that provide us with such functionalities as phones, TV, GPS are just one click far away from us. Anyone can satisfy curiosity for science and space easily.

In every screen during the search process, here are *BreadCrumbs* that helps the user know the satellite type picked and allows navigating through the type menus and English and Spanish are supported.

5. MINIMUM DEVICE REQUIREMENTS

ZATDROID has been developed and tested with:

- SDK: Eclipse
- Android Version: 2.3
- Device: Smartphone Sony Xperia Neo V (*GPS available, Wifi, 3G*)
- Screen: *480x854 pixels*, 265 *ppi*, 3,7'.
- Processor: Qualcomm 8255 de 1 GHz

Under these conditions, it is tested that every functionality works perfectly. But **compatibility** with other devices and versions needs to be explained:

- ANDROID: versions of Android from 2.3 up to 4.3 have been tested without any problem
- Devices: *LG, Sony, Nexus 4, HTC* seem to work as expected. It is with some *Samsung* devices where a bug has been noticed. In the Augmented Reality View there is a green filter as a layer and sometimes the arrow pointing at the direction of the satellite is not appearing. After some searching to solve it, no solution has been achieved, because the code is running right in every device except for some *Samsung* ones. *Samsung S3Mini* running ANDROID 4.2.1 has not this camera filter bug.
  Tablets with big screens show the windows fitting the size.
- Communications: WIFI or 3G internet connection needed.
- Tablets with no GPS and no phone capability cannot run the application.

6. INSTALLATION INSTRUCTIONS

As ZatDroid is created to run on smartphones or tablets with Android, the first step is getting a device like that with an active connection to the Internet.

There are many ways to get ZATDROID working. Basically, it consists of downloading the *apk* file and installing it in the device.

- By sending it via email, message or in sharing it in the cloud.
- By downloading it from Google Play. It is not at the moment uploaded, but it will be in a few weeks.
- By downloading the *apk* from free repositories like GITHUB, `commonsware.com.` It will be uploaded in some weeks.

Once the user has the *apk* file downloaded into your device, it is as easy as clicking on the installation button and accepting the permissions:

Permissions required when installing the *apk*:

- <u>Internal Storage</u>: To store XML and XSD files for searches
- <u>GPS Location:</u> to allow orbital mechanics
- <u>Network communication:</u> to allow orbital mechanics
- <u>Hardware controls (camera):</u> for the Augmented reality functionality

7. EXECUTING THE APP: SCREEN SEQUENCE DIAGRAM

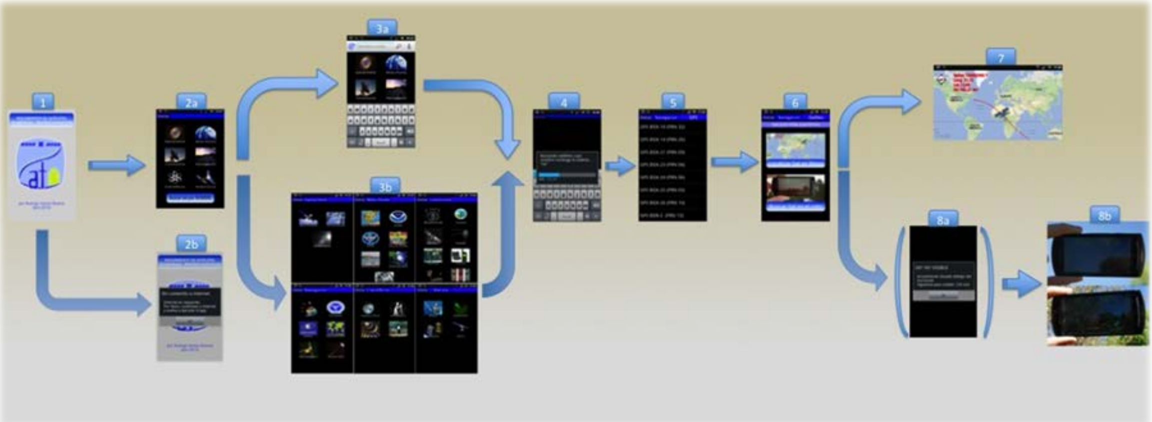


Figure 2. Screen Sequence Diagram

When executing ZATDROID, the users will be prompted several screens through the way. Figure 2 shows the general diagram sequence. This manual explains every screen and the processes followed behind the images.

7.1. INTRO

Figure 3. Screen 1: Intro

The first screen is showing the welcome with a brief description of the functionalities of the App. the logo, the author and the date. The logo includes a satellite at the top, the Android logo at the bottom and a big "Z" in the middle. Along the "Z" it is written the title of the App.

7.2. INITIAL SEARCH MENU

Figure 4. Screen 2a: Initial Search Menu

- This menu shows the two ways to perform a satellite search:
  - Through types pre-defined: there are icons with text, that will lead the user through a way of specific types until reaching finally a satellite.
  - By some key words: it opens a `searchDialog` to perform the search by key words.
- All the icons are adjustable to different screen sizes
- Supports Multilanguage: Spanish and English depending on the default of the device. (as shown in the images)
- *BreadCrumbs* navigation buttons helps the user.

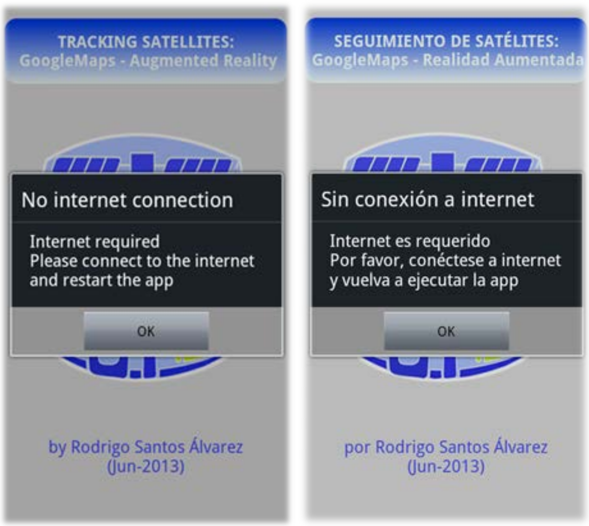


Figure 5: Screen 2b: Internet Connection Checking

Internet connection is needed, so if the device has no active connection, it shows the user a message and stops the application

7.3. SATELLITE SEARCH PROCCESS

Figure 6: Screen 3a: Search by key words

The option to carry out a search typing some key words is performed with the help of the `searchDialog`. Voice typing is allowed.

Figure 7: Screen 3b: Types of satellites

There are 6 screens describing the satellite subtypes. The screen structure is similar to the first screen, with *BredCrumbs* navigating buttons, adjustable icons size, Spanish and English support.

7.4. DOWNLOADING FILES FROM THE INTERNET

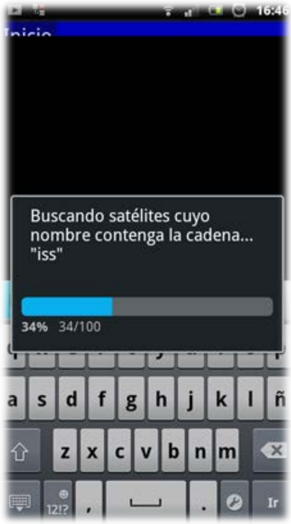


Figure 8: Screen 4: Downloading TLEs Progress Bar

After having picked the final satellite type or having typed the key words, a time-limited screen is shown with a progress bar to let the user know that the program is running. ZATDROID is getting the TLEs (*two line elements sets*) from `celestrak.com` with the information of the satellites in a txt file. Deep inside the code, txt file is transformed into a more easy-managed XML file with its XSD file linked.

The downloading process lasts more that a few seconds sometimes, depending on the internet connection. Actually:

- If the download is launched when a satellite type have been picked, it lasts just 1 or 2 seconds, because the txt file is just from a subtype.
- If the download is launched when a search by key words is performed, all the subtypes are downloaded and merged into one file. Therefore this process lasts more.

It is worth mentioning that the download process is performed using a ASYNTASK method, which allows the program to run in a second thread the download while the progress bar is shown to the user. If an error is found, the program does not stop suddenly. This ANDROID method increases the robustness of the software.

7.5. SATELLITE NAMES LIST

Figure 9: Screen 5: Listing satellite names from XML file

After the downloading process has finished, and then the XML file created from the txt with the satellite information and orbit parameters, only the names of the satellites are shown in a `ListView`. Also the XSD (*XML Scheme Definition*) is created to define the legal building blocks of the XML document.

Once the XML is created, a SAX (*simple API for XML*) search is performed to find the satellite names and be able to add them to the `ListView`. In case a search by key words is carried out, the list can be extremely large and also if the “Gestationary” subtype is picked. There are 414 satellites orbiting the Earth in the geostationary belt at this moment.

At the top, *BreadCrumbs* let the user know the satellite type and subtypes picked. The buttons are clickable so that the user can navigate through them.

When the user picks a certain sat name, another SAX search is launched to find the satellite in the XML file and extract all the information and parameters orbit from the file and use them to create a “*sat*” JAVA object with all that information. This “*sat*” object will be fulfilled with other orbit parameters calculated afterwards with orbital mechanics.

7.6. FUNCTIONALITIES MAIN MENU

Figure 10: Screen 6: Functionalities Menu

At the top, *BreadCrumbs* let the user know the satellite type and subtypes picked. The buttons are clickable so that the user can navigate through them. Here the full address is shown and the user can go back to any of the subtypes.

The “sat” Java object is retrieved but no orbital mechanics calculations is developed so far. The menus are composed by an image and a clickable button. The text supports English and Spanish, as usual, and the

- The first icon open the GOOGLE MAPS Activity where the sat will be represented together with its trajectory.
- The second button opens the Augmented Reality Activity, where the user can search the sky to find the satellite with the camera view.

7.7. GOOGLE MAPS FUNCIONALITY

Figure 11: Screen 7: GOOGLE MAPS Funcionality

Over the GOOGLE MAPS View, the satellite icon is positioned, together with its trajectory and the satellite orbit parameters: latitude, longitude and altitude. They are updated instantly, except for the trajectory, that lasts one second, due to the complicated calculations. The trajectory shows the next and past 20 minutes. The complete orbit cannot be drawn due to the big amount of calculations.

If the user ever sees a satellite at high altitudes (30.000 Km) and the trajectory is not drawn, it is not a bug. The satellite follows a geostationary orbit and therefore it is fixed. Actually, it is rotating with the Earth, so for an inhabitant it is always in the same position, same latitude and longitude.

Orbital Mechanics calculations are performed once this screen is shown, following NORAD (*North American Aerospace Defence Command*) SGP4 (*Simplified General Propagation Model Version 4*) / SDP4 (*Simplified General Deep Space Perturbation model version 4*) propagation methodology, according to NASA (*National Aeronautics and Space Administration*) documents.

7.8. AUGMENTED REALITY FUNCTIONALITY

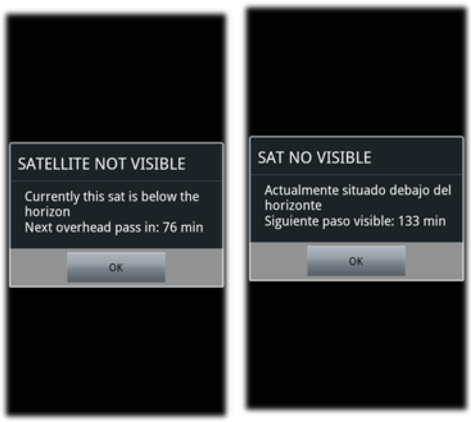


Figure 12: Screen 8a: Visibility Message

Figure 13: Screen 8b: Augmented Reality View

Finally, the last functionality is this Augmented Reality View.

- Orbital Mechanics calculations are developed to get azimuth and elevation of the satellite.
- OpenGL is used to match the camera view azimuth and elevation with the satellite position in the sky.
- Layers over the camera view are to set Text, arrows, central square and satellite icon.
- Georeference is needed to calculate the location of the device, latitude and longitude. GPS or network is used.
- Sensors like accelerometer, gravity and magnetic give the orientation of the camera view device.

If the elevation of the satellite from the device location is below 0 degrees means that the satellite cannot be seen over the horizon. In this case, a message is shown to inform the user and the time lasting until next overhead pass. (*English and Spanish supports, as shown*)

Orbital Mechanics calculations are again performed, following NORAD SGP4 / SDP4 propagation methodology, according to NASA documents.

8. TROUBLESHOOTING

- Some *Samsungs* devices (*Samsung S3, Note*) show a green filter in the Augmented Reality View over the camera view. However, *Samsung S3Mini* runs AR View as expected. Android versions do not influence and other devices have no problems like this. Some research has been done to fix it up, but nothing related have been found so far. The filter allows to see through the camera, so it is not really important.

- GPS or internet or network is a must when running ZATDROID. Tablets without these three features cannot execute the app and an error is shown. Device location (longitude, latitude and altitude) cannot be retrieved.

- Some *Motorola* and *Sony* tablets do not start the app. There is no explanation for this bug so far.

- If the internet connection is not quick enough, the downloads process could last more than 3 seconds as required. Anyway, download files when a search by key words is selected could last more than 3 seconds even with a quick connection.

- In the GOOGLE MAPS View , trajectory is refreshed every second, not instantly. The user notice that trajectory delays when moving or zooming. This is done on purpose, otherwise the big amount of calculations the app hangs.

In case other bugs are detected, please contact with the author. I will be grateful.