GEO-CLOUD: MODELLING AND IMPLEMENTATION OF THE GEO-CLOUD EXPERIMENT FOR THE FED4FIRE EUROPEAN PROJECT

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Geo-Cloud: Modelling and Implementation of the Geo-Cloud Experiment for the Fed4FIRE European Project

Rubén Pérez Pascual

September, 2014

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

Geo-Cloud: Modelling and Implementation of the Geo-Cloud Experiment for the Fed4FIRE European Project

Autor: Rubén Pérez Pascual

Director: Dr. Jonathan Becedas and Carlos Morcillo

September, 2014

Rubén Pérez Pascual

Ciudad Real – Spain

E-mail: Ruben.Perez@alu.uclm.es
Teléfono: 654 925 872
Web site:

This work was carried out with the support of the Fed4FIRE-project (“Federation for FIRE"), an Integrated project receiving funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 318389. It does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

TRIBUNAL:

Presidente:

Secretario:

Vocal:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	**SECRETARIO**	**VOCAL**
Fdo.:	Fdo.:	Fdo.:

Abstract

PROCESSING and distribution of big space data still present a critical challenge: the treatment of massive and large-sized data obtained from Earth Observation (EO) satellite recordings. Remote sensing industries implement on-site conventional infrastructures to acquire, store, process and distribute the geo-information generated. However these solutions do not cover sudden changes in the demand of services and the access to the information presents large latencies.

In this work we present the detailed design, architecture and implementation of the GEO-Cloud experiment to value if future internet technologies can be used to overcome the previously defined limitations.

The whole system implements a complete highly demanding EO system making use of future internet technology and cloud computing to define a framework to make EO industry more competitive and adaptable to the new requirements of massive data processing and instant access to satellite information.

Contents

List of Tables

List of Figures

Listings

List of acronyms

GNU	GNU is Not Unix
GPL	General Public License
LGPL	Lesser General Public License
BSD	Berkeley Software Distribution
SDL	Simple DirectMedia Layer
RPC	Remote Procedure Call
XML	eXtensible Markup Language
CSW	Web Catalogue Service
DBMS	Database Management System
API	Application Programming Interface
EaaS	Elasticity as a Service
SQL	Structured Query Language
OML	Object Monitoring Language
OMF6	Object Monitoring Framework v6
XMPP	Extensible Messaging and Presence Protocol
SQL	Structured Query Language
EO	Earth Observation
LTAN	Local Time Ascending Node
LEO	Low Earth Orbit
FP7	Seventh Framework Programme
OGC	Open Geospatial Consortium
QoS	Quality of Service
ISO	International Organization for Standardization
SRTM	Shuttle Radar Topography Mission
USGS	U.S. Geological Survey
NASA	National Aeronautics and Space Administration

SaaS	Software as a Service
IaaS	Infrastructure as a Service
WMS	Web Map Service
WFS	Web Feature Service
WPS	Web Processing Service
WCS	Web Coverage Service
EaaS	Elasticity as a Service
PaaS	Platform as a Service
SSO	Sun Synchronous Orbit
ICMP	Internet Control Message Protocol
UDP	User Datagram Protocol
TCP	Tranport Control Protocol
SSL	Secure Sockets Layer
IP	Internet Protocol
GUI	Graphical User Interface
GSD	Ground Sample Distance
RPC	Remote Procedure Call
SSH	Secure SHell
CORBA	Common Object Request Broker Architecture
SOAP	Simple Object Access Protocol
ICE	Internet Communications Engine
JSON	JavaScript Object Notation

Acknowledgements

Escribe aquí algunos chascarrillos simpáticos. Haz buen uso de todos tus recursos literarios porque probablemente será la única página que lean tus amigos y familiares. Debería caber en esta página (esta cara de la hoja).

Rubén

A mi familia, a Carlos que confió en mí y me dió a conocer y a mis compañeros de Deimos Castilla-La Mancha, especialmente a mis compañeros espartanos de I+Di.

Chapter 1

Introduction

EARTH Observation (EO) commercial data sales have increased a 550% in the last decade [Mar08][1]. This area is considered a key element in the space industry and an opportunity market for the next years.

EO industries implement on-premises conventional infrastructures to acquire, store, process and distribute the geo-information generated.

However these solutions have the risks of over/under size the infrastructure, they are not flexible to cover sudden changes in the demand of services and the access to the information presents large latencies. These aspects limit the use of EO technology for real time use such as to manage crises, natural disasters and civil security among others (Deren, 2007).

In addition, new sectors and user typologies are applying for new EO services and there is an increasing demand of this services. These users need more flexible, easy and instant access to EO products and services through the Web. This demand has traditionally been driven through Space Data Infrastructures and heavy standards (ISO TC/211 and Open Geospatial Consortium (OGC)) which are focused on interoperability rather than the real demand from the end-users.

The use of cloud computing technology can overcome the previously defined limitations that present conventional infrastructures because of its elasticity, scalability and on-demand use characteristics (Armbrust, 2010).

GEO-Cloud Experiment goes beyond conventional data infrastructures used in EO industry and beyond the implementations of applications running in cloud, to quest which parts of a complete infrastructure of EO are technologically and economically viable to be virtualized to offer basic and high added value services (see Figure 1.1).

GEO-Cloud emulates the remote sensing mission with the satellites, the topology network and the communications in the *Virtual Wall* testbed. The data acquired from the emulated satellites is transferred to the *BonFIRE* cloud for storage, processing and distribution of data. End users accessing and broadcasting will be emulated in another network implemented in *Virtual Wall* . In order to implement realistic impairments in *Virtual Wall* , real networks will be tested in *PlanetLab* . The technologies for imagery distribution and EO service delivery

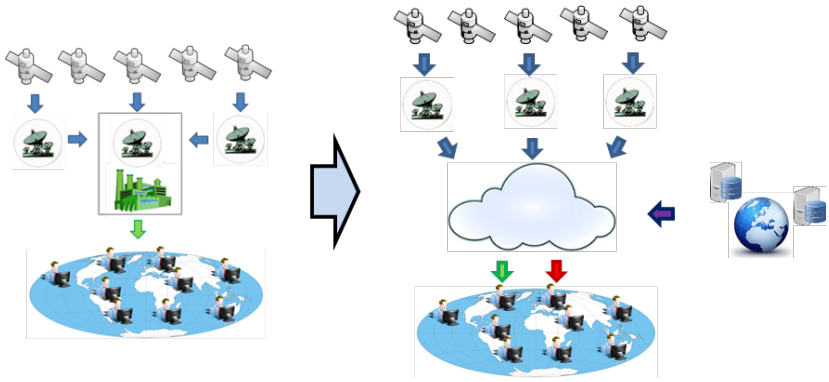


Figure 1.1: The GEO-Cloud Concept. It is based on the use of cloud technology to acquire data, store it, process it, integrate it with other sources and distribute it to end users with the final objective of testing viable solutions for its real implementation.

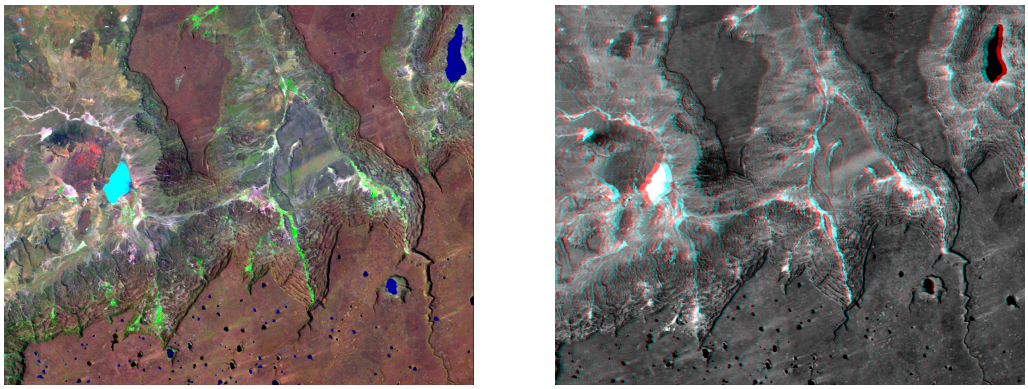
using cloud technologies and Internet protocols will be tested.

1.1 Earth Observation

Since decades, the human's ambitions consists of knowing themselves and everything outside them. However, technology of that time did not permit these aims. In last decade, quite technological advances have been carried out and these goals are the present. Spacecrafts, airplanes, and several technology elements aggregated by a space mission are designed, built and launched to retrieve information about the Earth, other planets and galaxies.

The EO involves all current technical areas such as Computing, Optics, Chemistry, Mathematics, Materials, Telecommunications, Aerospace, Physics, System Engineering among others. The physical, chemical and biological information about the Earth are gathered using several ways for obtaining the information and platforms to achieve it.

The Earth surface information can be achieved both actively and passively. Active remote sensors as radar emit microwaves toward the Earth's surface in order to scan objects and areas. These waves reflect off the surface and return to the sensor. This imaging way is also known as active microwave and three types of actively remote sensing are: imaging radar, that takes images like a camera depending on the type of surface where the rays reach; non-imaging radar that measures the amount of reflected energy; and altimetry sensor which sends microwaves to Earth and measures the time that these microwaves take to return the sensor.
Passive remote sensors use the radiation provided by the objects or surrounding areas. The most common source for gathering passively, is the reflected sunlight.
Passive remote sensors are radiometers for measuring the electromagnetic radiation, photography to acquire the light reflected by the chemical elements in visible spectral band, charge-coupled devices catching ultraviolet spectrum and infrared sensors which obtain thermal images. Moreover, the EO acquisition takes into account three kinds of resolution for



(a) Image acquired in blue, near infrared and short wave infrared spectral bands.

(b) Image acquired from SRTM conforming a 3D image.

Figure 1.2: Different images acquired by USGS/NASA Landsat.

imaging as: the spectral , the geometry and the temporal resolution. The spectral resolution depends on the object to be sensed. Panchromatic, blue, green , red, near-infrared and thermal infrared are the most used spectral bands.

The geometry resolution is selected depending on how much definition the telescopes or imaging sensors has to be. If a cloud shall be acquired for weather applications, the resolution will be about $100km$. In other hand, if the application is for traffic management, the telescope resolution must be under $1m$.

The temporal resolution is the frequency between two consecutive acquisitions of the same location (revisit time). Depending on the image application, the revisit time can vary. For example, for environmental monitoring, geology or precision agriculture the revisit time is longer that the required revisit time for military or maritime surveillance, that in these cases is critical.

The last component for EO imaging consists of the platform where the payloads for image acquisition are onboard. These platforms are summarized in two sets, spacecrafts and aircrafts. Aircrafts involve aerodines (planes, and helicopters among others) and aerostats (balloons and dirigibles). Spacecrafts are systems designed to fly over the atmosphere and could be used for lots of purposes as communications, EO, meteorology and planetary exploration among others.

1.2 Cloud Computing

Traditional business applications have always been too complicated and expensive. The number and variety of necessary hardware and software to run them is overwhelming. A team of experts that can install, configure, test, run, secure, and update is needed. Thanks to Cloud Computing approach, these complications do not exist because it is not necessary to manage the hardware and software: it is the responsibility of an experienced cloud provider.

The shared infrastructure makes it work like a utility: You only pay for what you need, upgrades are automatic and the enlargement or reduction of the service comprises a simple process.

The NIST **??** defines *Cloud Computing* as "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

Several cloud deployments are implemented at present. These are:

- *Private Cloud:* Used by a single organization and it is operated by the own organization.
- *Community Cloud:* Organizations that have shared subjects create this kind of cloud in which the managed and support is carried out by the organizations themselves.
- *Public Cloud:* It is provisioned for the general public use it. The management of the infrastructure is performed by an academic, business or government organization.
- *Hybrid Cloud:* Cloud composed by distinct infrastructures as public, community or private in order to achieve some objectives.
- *Federated Cloud:* Nowadays, this deployment is growing up quickly. The federated architectures are a combination of community, public and hybrid clouds in which each one of them offer features that the consumer can be used transparently. In the section 2.4.2 are detailed.

The cloud computing features are summarized as follows:

- *On-demand self-service:* A cloud computing user can provision, deploy and release computing resources as needed automatically without human interaction.
- *Broad network access:* Cloud capabilities can be accessed by standard mechanisms or client platforms.
- *Resource pooling:* Different resources can be assigned as memory, processing capability, network bandwidth among others, and these are distributed geographically or into several cloud machines.
- *Rapid elasticity:* Resources can be elastically provisioned and release automatically. This permits to scale rapidly resources.
- *Measured Service:* Resources can be monitored, controlled and reported in order to achieve transparency for the cloud supporter and consumer.

The cloud computing infrastructure offers different kinds of services that are summarized as:

- *Software as a Service (*SAAS*):* The capacity provided to the consumer to use the provider's applications running on the cloud.

- *Platform as a Service (*PAAS*):* The capacity provided to the consumer consists of the consumer-created applications created by languages, libraries and tools supported by the provider can be deployed on the cloud.

- *Infrastructure as a Service (*IAAS*):* The capacity provided to the consumer consists of provisioning computing resources in order to the consumer is able to deploy and run its developed software.

1.3 The Fed4FIRE European Project

The *Fed4FIRE* is an Integrating Project under the European Union's Seventh Framework Programme (FP7) which it work programme topic is *Future Internet Research and Experimentation*. The project is performed by a consortium of 29 partner from different countries. The project is coordinated by the *iMinds*, Belgium. The *Fed4FIRE* project establishes a common federation framework for experimenters. A large of number of facilities in Europe are integrated in the *Fed4FIRE* federation. Such facilities focus on different areas of networking. Example domains are wireless networking, cloud computing, smart cities and grid computing among others. Also, the *Fed4FIRE* project has to validate its infrastructure to perform innovative experiments, so researches from different Future Internet areas are invited. As a result, projects like *GEO-Cloud* are carried out.

1.4 The Geo-Cloud Experiment Overview

1.4.1 Experiment Description

The experiment consists of virtualizing a conventional EO system to offer on demand services to clients with the objective of validating its viability, find the strengths and weaknesses of using cloud computing technology and establish possible solutions for a future implementation in the market. There are three components:

1. *In-orbit mission:* this component generates the raw data. This consists of un-processed images of the Earth captured by a constellation of satellites and downloaded to different ground stations.

2. *Treatment of data:* the data has to be stored, processed at different levels based on the services offered and distributed to the clients. The data acquired by the in-orbit mission is integrated with other sources to provide higher quality services.

3. *End-users:* users of the provided services with different levels of remote access rights.

1.4.1.1 Experiment Design

The GEO-Cloud experiment requires emulating a complete realistic Earth Observation Mission to provide high added value services such as crisis management. To this complex situation, the system has to response by processing on demand massive and variable amounts of stored and on line transferred data.

GEO-Cloud makes use of the following *Fed4FIRE* facilities: *PlanetLab* ,*Virtual Wall* and *BonFIRE* . *PlanetLab* allows us to measure real network characteristics geographically distributed to setup our models. *Virtual Wall* allows us to create any desired network topology and emulate the in-orbit mission and the web service to the users. *BonFIRE* provides us a real cloud infrastructure with observability in all the layers to test our cloud based services.

1.4.1.1.1 Implementation of the acquisition of geo-data in Virtual Wall and PlanetLab

The acquisition of geo-data is obtained from the in-orbit mission. The constellation of satellites and the ground stations are emulated in *Virtual Wall* . A network topology is implemented to communicate the different satellites with the ground stations. Every satellite in its orbit and every ground station models are simulated in a node.

The satellite models simulate the orbits and the pass of the satellites over the ground stations. The ground stations models simulate the coverture of the antennas and the download of the data. When a satellite is inside this radius, the satellite downloads the data to the ground station that is visible. The downloaded data in the ground stations is transferred to the *BonFIRE* cloud.

With the *Virtual Wall* network,the *bandwidths, latencies and loss rates* are controlled.Also a realistic network topology to transfer data between different nodes is created.

In order to determine the correct link characteristics for the connections between the ground and the cloud infrastructure, a profiling tool has been developed for measuring appropriate values for the link impairment between these different geographical locations using the *PlanetLab* testbed.

1.4.1.1.2 Implementation of the cloud based services in BonFIRE

To facilitate offering the previous services we propose to implement a multi-layered cloud model in the *BonFIRE* cloud infrastructure to generate on demand geo-information. The multi-layered cloud model is constituted of two layers:

- *Layer 1:* This layer involves the basic satellite imagery services. It acquires the raw data, stores it, has the first level of processing, distributes the processed data and offers the hosting service.

- *Layer 2:* This layer involves the high added value services. It can use historical processed, real time captured and pre-processed data from layer 1. This layer processes the information for real time generation of geo-information and offers real time access and distribution to the end-users. Typically, the implementation of high added value EO services involves the ingestion of the raster imagery from the satellites into a spatial database or storage, where it can be refined, simplified, processed or combined with other data sources in vector or raster format. The products, which can be vector or raster data, are distributed or queried using Internet technologies (OGC standards like Web Map Service (WMS)) or through Web services (tiles, caches, etcetera).

Thus, the whole EO system is completely implemented in *Fed4FIRE*, see Figure 3.

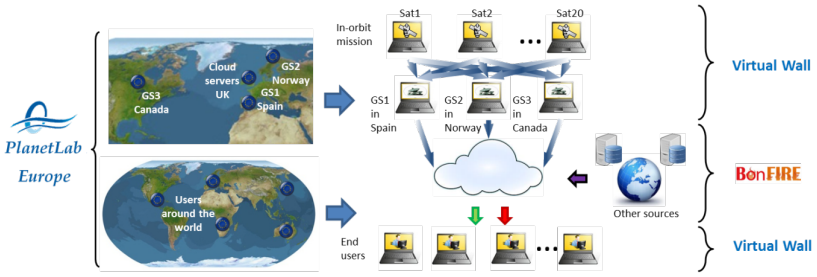


Figure 1.3: Geo-Cloud implementation in Fed4FIRE

1.4.2 Impact in Fed4FIRE

The experiment will contribute to several objectives of the *Fed4FIRE* project: increase trustworthiness of its facilities and support their sustainability. GEO-Cloud will test *Fed4FIRE* tools for its use in the industry driven experiments close to market, specifically complex and real time services for Earth Observation industry when critical situations occur. GEO-Cloud will validate the tools for monitoring and control of cloud computing and networking in EO services for emergencies and will test the limits of the infrastructure for processing, storing and traffic of massive on-demand data. GEO-Cloud will provide feedback to improve the infrastructure during and after the experiment is carried out, sharing our knowledge in traditional infrastructures for EO applications, monitoring, processing and distribution of geospatial data.

1.4.3 Scientific and Technological Impact

GEO-Cloud will contribute to provide a worldwide service in the Earth Observation Industry. It will answer if Future Internet technologies can provide viable solutions for the complex EO market and to find the limitations of the current cloud computing technology for its application in EO market.

GEO-Cloud will test the viability and ease of use of those facilities for industry driven experiments close to the market.

1.4.4 Socio-Economic Impact

The Geo-Cloud project is used as a framework to offer services from EO users. The benchmark developed in the experiment allows to establish the frontiers of viable and not viable cloud solutions in EO depending on the type of demand and service offered. This will establish the basis to satisfy the growing demand of added value EO services.

The reduction of the processing, storage, communications and distribution costs of EO services will facilitate the access to the remote sensing technology of common end users, but also of a more general public. GEO-Cloud will contribute defining the basis to advance in the use of geospatial information of the nine "Societal Benefit Areas" defined in GEO: disasters, health, energy, climate, water, weather, ecosystems, agriculture and biodiversity; by demonstrating whether or not cloud computing offers technologically and economically viable solutions to offer highly demanding services.

The results obtained in the experiment will be used by our company to offer new services to the general public, current end users and future potential users. The rest of the EO industry will be beneficiated since we will define a benchmark that relates the demand with the technology to offer quality services. The users will be beneficiated since we will define the framework to offer higher quality services.

1.5 Document Structure

This document has been carried out as the end of carrier project rules from the *Escuela Superior de Informática* of the *Universidad de Castilla La Mancha*. It contains the following sections:

Chapter 2: Project Background

In this chapter an overview of the necessary knowledge areas is made to study for the development of GEO-Cloud, like cloud computing, distributed middleware, processing premises for EO and several testbeds in *Fed4FIRE*.

Chapter 3: Objectives

In this chapter, the main objectives for GEO-Cloud are depicted and explained.

Chapter 4: Method of work

in this chapter the selected methodology is explained and justified. Also, the uses resources such *Fed4FIRE's* testbeds, hardware and software are depicted.

Chapter 5: The Geo-Cloud Experiment

In this chapter, the entire GEO-Cloud project is explained. The design and implementation of the satellite constellation, the design and implementation of the satellites and ground stations simulators over *Virtual Wall* , the design and implementation of the cloud architecture for EO and finally the *PlanetLab* experiment for acquiring network impairments are detailed.

Chapter 6: Project Evolution and Cost

In this chapter, the project evolution during the development detailing the phases and iterations besides the inconveniences and decisions to solve them are described. Also, costs of project and schedule are shown.

Chapter 7: Results

In this chapter, the results of the development of the project are shown.

Chapter 8: Conclusions

In this chapter, the development conclusion and the reached objectives are summarized. Some future work and lines of research are suggested.

Chapter 2

Project Background

This chapter resumes the main subjects acquired during the accomplishment of the GEO-Cloud project. The novel architecture proposed in this project for EO processing carries the absence of this kind of platforms to compare with. In this way, the study of the state of the art will be done focalising in some aspects of cloud computing platforms, federated testbeds, networking, satellite systems, and so on. As consequence, this section depicts the different thematic areas . The conceptual map showed in Figure 2.1 presents the sections and subsections of this chapter. Thereby in Earth Observation Satellites area, the different concepts for creating and modelling a satellite for earth imaging, its data management and orbital definitions is studied. In high-level languages, some scripting languages are compared. In Networking section, different network impairments and several tools for acquiring these are depicted. In Federated Infrastructure section, the testbeds that Fed4FIRE platform is composed by are explained.

In the section that explains the Graphical User Interfaces background, several frameworks for developing GUIs are depicted. In Database section, different database management systems are compared. The distributed systems section explains and compares some middleware as Hadoop or ZeroC Ice for developing distributed applications. Finally, in the Software Design section discusses the requirements for building multiplatform code, and other ancillary libraries required by the detailed objectives of GEO-Cloud.

2.1 Earth Observation Satellites

In this section, different areas of aeronautics are involved. Orbital Mechanics which defines and calculates the satellite orbits for rounding around the world; Spatial Telescopes which contributes with the payload parameters as Ground Sample Distance (GSD) and Swath among others; finally, the data management which mission consist of processing on fly, storing and sending the science and ancillary data from satellite to Ground Segment.

2.1.1 Orbital Mechanics

This science [**?**] studies the motion of the satellite considered as a point with no mass in space which is affected by *Newton's Laws of Motion*and *Newton's Law of Universal Gravitation*. *The Newton's Law of Universal Gravitation* says that *any two bodies in the universe*

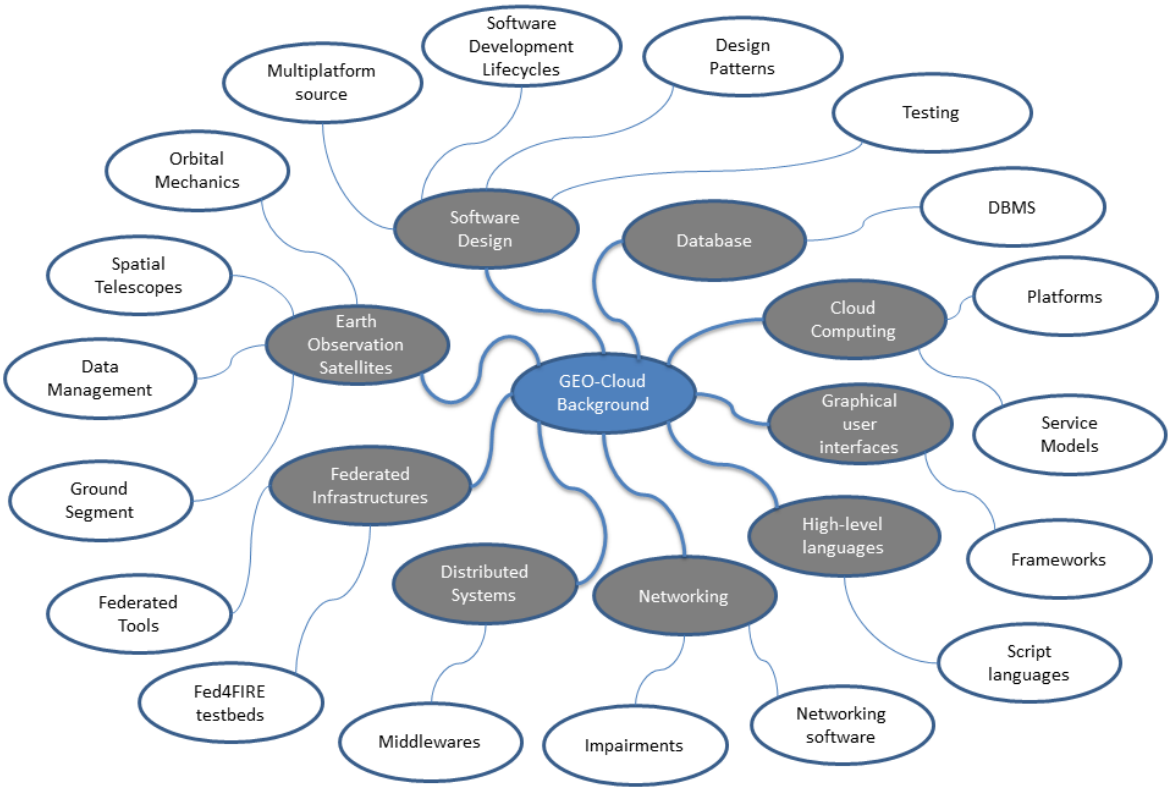


Figure 2.1: Conceptual Map of this chapter.

attract each other with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them.

$$F = G * \frac{m_1 * m_2}{r^2} \tag{2.1}$$

where F is the force between the masses, G is the gravitational constant ($6.67x10^{-11}$), m_1 and m_2 are the first and second mass respectively and r is the distance between the centres of the masses.

Orbital Mechanics focuses on the trajectories of the spacecrafts, manoeuvres for orbital acquisition, orbit maintenance, end of life disposal, etcetera.

Spacecrafts motion is governed by the *Kepler's Laws of Planetary Motion* which can be derived from *Newton's Laws*. These laws are the following:

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

major axis of its orbit.

2.1.2 Spatial Telescopes

The EO satellites carry on board telescopes pointing to the Earth in order to acquire images of the surface. The information can be obtained in several kinds of spectral bands as visible, near infra-red, thermal, etcetera. This mission employs visible multispectral telescopes distributed in a constellation of satellites to obtain daily a global map. For selecting the telescopes, the Swath and GSD have been analysed. The number of satellites is dependent on the width of the swath and the telescope resolution is required to achieve quality images for the mission.

Depending on the spectral bands employed for imaging, the applications are different. Scientific applications as soil categorization, vegetation analysis, oceanography use hyperspectral imagers which offers information about hundreds of bands.

Other applications as traffic monitoring, urban development, surveillance commonly require multispectral imagers with visible bands.

2.1.3 Data Management

Each satellite of the constellation acquires images which have five spectral bands into the visible spectrum. When this spectral data is obtained by telescopes, it is necessary to convert from analogical to digital data. This process is carried out using 12 bits per pixel for codifying the intensity signal level with a digital value. Then, the digital data is stored and pre-processed on board.

During the pre-processing activity, some meta-data that is well known as ancillary data, is added. The ancillary data enriches the acquired sector of image by adding some geolocated information and transmission auxiliary meta-data for communication protocols.
All the above pre-processed data is aggregated in the internal storage for downloading when the satellite comes into a visibility zone of a ground station. At the moment the satellite enters in a footprint of a ground station, the acquired images at real time and the stored images are downloaded into ground station.
Sending data is made multiplexing the communication channel within two ways. The first part of the total bandwidth is used for downloading the acquired images at real time and the other part of the rest of bandwidth performs the download of the stored images into the internal storages.

2.1.4 Ground Segment

The Ground Segment is composed by the antennas, the control centre and the infrastructures for processing and distributing images named processing data centre.

The current industries of EO imaging perform the processing, distributing, and selling on

premises. The steps are the following:

1. First, the antennas receives all the data from satellites and store it.
2. Once the data is into the antenna, the processing data centre obtains the images from the ground stations.
3. The data centre starts to process available images.
4. When a image has been processed is sent for archiving and cataloguing.
5. Finally, the image is available for end-users.

The architecture on-cloud proposed in this project, reduces the delivery end-user time making shorter the cataloguing and archiving times.

The control centre manages the satellites, schedule mission planning and collects telemetry but not interferes in downloading nor processing images.

In order to implement the archive and catalogue module in cloud, several platforms for sharing geospatial data were studied. These software servers are *GeoServer* and *GeoNetwork* among others.

- *GeoServer:* is a Java-based software server for cataloguing and archiving geospatial images among others functions, that uses OGC standards and allows users to view geospatial images. By default it offers Web Feature Service (WFS) services and Web Processing Service (WPS). In GeoCloud, the Archive and Catalogue module has been implemented using *GeoServer* in assembly with its Web Catalogue Service (CSW) extension, because it is flexible, simply, multiplatform, open source and it can receive data from other sources. Also, it can be integrated with *GeoNetwork*.

- *GeoNetwork:* is a Java-based and platform independent catalogue application to manage geospatial images. It uses OGC standards as CSW, WMS and Web Coverage Service (WCS) among others interchanges protocols. It catalogues data from sources but it not maintains its data. This is the reason for *GeoServer* was selected.

2.2 High-level languages

For the development of the project, eXtensible Markup Language (XML), JavaScript Object Notation (JSON),*Python* and *Bash* languages has been used. The first, XML, has been used for building the configuration file of the Orchestrator component and to obtain the selected nodes by *JFed* application. Then JSON has been used to create the experiment descriptor for *BonFIRE* platform. The rest of them are scripting languages. The source of the components of the cloud has been developed using *Python* and the interconnections between modules and others secondary functionalities, in *Bash script*.

2.2.1 XML

XML is a mark-up language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. Most of actual software use it for configuring or for updating its configuration on fly.

2.2.2 JSON

JSON is a lightweight language for data interchange between applications. The simplicity of JSON results very simple and human-readable way to transmit data objects consisting of attribute-value pairs. Web applications are using this language substituting XML because parsing and generating JSON is more efficient and quick. The *BonFIRE* interface uses it for creating experiment descriptors ought to the resources provided by the platform can be translated into objects and its features.

2.2.3 Scripting languages

The Scripting technics consist of use a interpreted programming language in order to provide advanced mechanisms for specifying functionalities of an application. The most important features of a scripting language are that is not compiled and it permits effortless development. Also, some interpreted languages are multiplatform because is not necessary to compile the source for running in target platform.

2.2.3.1 Python

https://www.python.org/ *Python* is a object oriented language, although permits imperative and functional programming. It does not need to compile the source because it is interpreted. It permits a flexible development because it is dynamic type and in this project, the most important feature why this language has been used, consist of it is multiplatform. With this feature, the software is able to be executed in any platform. The current version is 2.7.4 and it offers several useful and multipourpose libraries as *matlib, pthread, mysqllib, pdb* and so on.

2.2.3.2 Bash Script

Really, *Bash* is not a language, is a command language interpreter for the GNU operative system. *Bash* is fully compatible with other command language interpreters like *sh* or *ksh*, so the source developed by *Bash*, normally, is portable. However, the *Bash* language is also known as *Bash*. The Bash scripts contains commands for being executed by the interpreter. This way provides a direct line to communicate with the operative system and to make some functionalities ad-hoc. The *Bash* language consists of several sentences that the operative system is able to read and translate in order to play a specific action. All the currents *Linux* distributions contains a Bash interpret.

2.2.3.3 Ruby

https://www.ruby-lang.org/en/ *Ruby* is an object-oriented, cross-platform, general-purpose and dynamic programming language. It was designed and developed by *Yukihiro Matsumoto*. This language has a dynamic type influenced by Groovy, Falcon and other ones. It permits an easy, flexible and agile development.

2.2.3.4 Lua

http://www.lua.org/ *Lua* is a lightweight,cross-platform, prototype-based, object-oriented programming language. It was designed and developed by *Roberto Lerusalimschy*. This language is influenced by C++, Modula and Scheme among others. It provides native data structures as tables, records and associative arrays which this structure perform high throughput.

2.3 Networking

For simulating an environment as real as possible in GEO-Cloud, the impairments of the networks between both Ground Stations and Cloud Platform and between both Cloud Platform and end-users had to be acquired. Also, features like bandwidth is obtained in order to stablish the maximum throughput of channel over *Virtual Wall* .

There are several simulators that calculates the value of these network features roughly but the obtained results may be wrong. Therefore, a sub-experiment explained in Section 5.7 for acquiring these values is implemented.

2.3.1 Impairments

The communication though the Internet is not perfect because there are many sources or random events that affect traffic packets that are traversing a network differently at any given moment. This is a main subject to take into account when a network is being modelling. The main impairments of a network are the following:

- *Packet Loss:* This is simply the disappearance of a packet that was transmitted or ought to have been transmitted.
- *Packet Delay:* Also known as “Latency” is the amount of time that elapses between the time a packet is transmitted to physical environment until the packet is received by target. This delay is composed by three types of delay times: Propagation delay which is the time that the packet waste arriving destination; Routing/Switching delay that is the time that the routers or switches waste in processing the packet; finally the Queuing delays that is the time that the packet is queued in any intermediate hardware of the entire network.
- *Jitter:* Jitter is a measure of the variation in the packet delay experienced by a number of packets.

- *Packet Duplication:* May occurs when one packet become two or more identical packets.
- *Packet Corruption:* It occurs when the payload of the packet (even a bit) is damaged but the packet continues to flow towards the destination instead of being discarded.

For GEO-Cloud project, the impairments to be acquired are *Packet Loss* and *Packet Delay* that permit modelling a simulated and close to reality network.

2.3.2 Networking Software

In the networking subject, there are lots of tools that permit to obtain the features values for the metrics defined above in a network. For measuring the throughput, bandwidth and loss-rate the following tools are interesting:

- *Microsoft's NTttcp http://gallery.technet.microsoft.com/NTttcp-Version-528-Now-f8b12769:* Test tool is effectively iperf on steroids and optimized for Windows environments. NTttcp does one better in correlating in CPU usage for a network task as well as allowing mapping to CPUs for systems with multiple processor capability.
- *NetCPS:* Is an oldie but a goodie, and between iperf and NTttcp, most basic network testing can be performed to give a good baseline of network performance.
- *Iperf https://iperf.fr/:* the most popular tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay, jitter and datagram loss. JPerf is a Java-extension for providing a graphical user interface.
- *Uperf http://www.uperf.org/manual.html:*Is a network performance measurement tool that supports execution of workload profiles. It is more complex and complete than Iperf or NetPerf allowing the user to model an application using a very high level language and running this over the network. It allows to use multiple protocols, varying message sizes, to collect statistics among others.
- *NetPerf http://www.netperf.org/netperf/:* Is a benchmark that can be used to measure the performance of many different types of networking. It provides tests for both unidirectional throughput, and end-to-end latency.

For measuring the delay time the following tools are depicted:

- *Ping:* It is the most used tool for testing the reachability of a host on an Internet Protocol network and for measuring the round-trip-time for packets sent from a source host to a target host. The sent packets are Internet Control Message Protocol (ICMP) protocol.
- *Traceroute:* Is a networking tool for displaying the path and measuring transit delays of packets across a network over the Internet Protocol. This command is available on

a number of modern operative systems. By default it works in layer 3 sending User Datagram Protocol (UDP) packets but it can be customized for sending ICMP packets.

2.4 Federated Infrastructures

In this section, some federated infrastructures are depicted. This infrastructures are used by experimenters for creating new network topologies, new distributed applications, new network protocols, among others.

A federated infrastructure is a set of unified and autonomous platforms which are joined in order to provide interoperability and coordinated information sharing among individual components.

The federated architecture pattern was first used by the *US Federal CIO* in 1990s. Then other organizations adopted this paradigm for its infrastructure technologies. Nowadays, this topology is growing up in business where the technological infrastructure has to be distributed and they must share critical information around the world.

The benefits of this kind of network topology are enumerated as follows:

- Independence: the components of the federation has its own rules, protocols, subcomponents, and so on. All of them provides an interface for communicating among others federation components.
- For the end-users the federated cloud provides an easily way to host apps and the automatically selection of resources from different federated components.

There are several federated infrastructures for experimenting. These are known as "Federated testbeds" and the most important among others are the following:

- *GENI*: The *Global Environment for Networking Innovation*, is a distributed virtual laboratory for the future internet sponsored by the *U.S. National Science Foundation* for experimenting. In this platform may be carry out developments like protocol design and evaluation, distributed services, content management and experiments that needs wide areas[1].
- *Open Cirrus:*The *Open Cirrus* testbed provides a federation in cloud computing for experimenting. To support these experiments, global services has been added for providing distributed common services. The experiments that can be carried out in this testbed are large-scale clustering experiments, machine learning, scientific computing among others. *Open Cirrus* is composed by 10 sites in North America, Europe and Asia and the platform has been developed by the *U.S. National Science Foundation*, *the University of Illinois*, *the Karlsruhe Institute of Technology*, *the Infocomm Development Authority of Singapore*, *the Russian Academy of Sciences*, *the Electronics and*

[1]For more information, see `http://www.geni.net/`

Telecommunications Research Institute of South Korea, *the Malaysian Institute of Microelectronic Systems* and *Carnegie Mellon University*. This project is sponsored by Hewlett-Packard, Intel and Yahoo! [2].

- *Fed4FIRE:* The *Fed4FIRE* is an Integrating Project under the European Union's FP7 addressing the work programme topic Future Internet Research and Experimentation. The project is performed by a consortium of 29 partners organisations from 8 countries. The *Fed4FIRE* proposal consists of to create a heterogeneous and federated platform with different kinds of services and applications such cloud computing, grid computing, smart cities, wireless networking and large-scale experiments. The federation is composed by the following testbeds:
  - *Virtual Wall:*For creating network topologies.
  - *PlanetLab Europe:* For experimenting with nodes around the world.
  - *Norbit:* For experimenting with Wi-Fi resources.
  - *w-iLab.t:* This testbed is intended for Wi-Fi and sensor networking experimentation.
  - *NETMODE:* It consists of Wi-Fi nodes connected with some processors for networking experimenting.
  - *NITOS:* Is a testbed offered by NITLab and consists of wireless nodes bases on open-source software.
  - *Smart Santander:* This is a large scale smart city deployment in Santander city of Spain to experiment Internet of Things.
  - *FuSeCo:* This testbed provides resources to experiment with 2G,3G and 4G technologies.
  - *OFELIA:* For testing and validating research aligned with Future Internet Technologies.
  - *KOREN:* It provides programmable virtual network resources with necessary bandwith connecting 6 large cities at the speed of 10 Gbps to 20 Gbps.
  - *BonFIRE:* It is a multicloud tesbed for experimenting.
  - *performLTE:* Is a realistic environment for creating experiments using the LTE technology.
  - *Community-Lab:* It is a distributed infrastructure for researchers to experiment with community networks for creating digital and social environments.
  - *UltraAccess:* It provides several Optical network protocols and resources to experiment with Quality of Service (QoS) features, traffic engineering, virtual LANs and so on.

[2]For more information, see `http://opencirrus.org`

In GEO-Cloud project, the testbeds used for implementing are *Virtual Wall* ,*PlanetLab* and *BonFIRE* . These facilities are explained in the next sections.

2.4.1 Fed4FIRE Testbeds

In the section above the *Fed4FIRE's testbeds* are numerated. Now, the facilities that are employed in GEO-Cloud are detailed.

2.4.1.1 Virtual Wall

Virtual Wall is an emulation environment for experimenting with advances networks, distributed software and service evaluation carried out by the *University of Ghent*. It offers the possibility for experimenters to create any type of network topology, e.g. emulating a large multi-hop topology, client-server topologies among others and to check algorithms or protocols for subsequent marketing. Two *Virtual Wall* are available: *Virtual Wall* 1 contains 200 servers: 100 quad-cores and 100 eight-cores; *Virtual Wall* 2 contains 100 dodeca-cores.

All servers have a management interface and five Gigabit Ethernet connections interconnected all of them by switches. On each of these links,the network impairments as bandwidth, loss-rate and delay can be configured. Also there are some virtual machines for customising the resources as the experimenter wants. As last, the network impairments as bandwidth, loss-rate and delay are configurable.

2.4.1.2 PlanetLab Europe

PlanetLab Europe is part of the *PlanetLab* global system, the world's largest research networking facility, which gives experimenters access to Internet-connected Linux virtual machines. About 1000 servers conform *PlanetLab* platform located in United States, Europe, Asia and elsewhere as Figure 2.2 depicts. This platform can be used by experimenters in order to develop and to check distributed systems, network protocols, peer-to-peer systems, network security and network measurements among others applications.

Through *Fed4FIRE*, the experimenters are able to reserve and deploy some PlanetLab Europe resources and to experiment with them.

2.4.1.3 BonFIRE

BonFIRE is a multi-cloud testbed based on an IAAS delivery model with guidelines, policies and best practices for experimenting. Currently, *BonFIRE* is composed by 7 geographically distributed testbeds, which offer heterogeneous cloud services, compute resources and storage resources.These testbeds are *EPCC*, *INRIA*,*Wellness*, *HLRS*, *iMinds* and *PSNC*. The Figure 2.3 shows these testbeds and its interconnections.

The compute resources that *BonFIRE* offers are summarized in Table 2.1.

The setup of compute resource can be done using contextualization variables in order

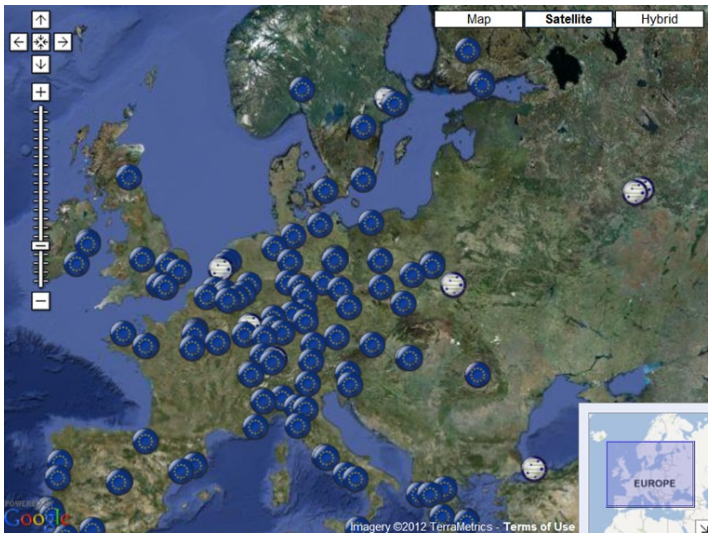


Figure 2.2: Geographical distribution of PlanetLab Europe.

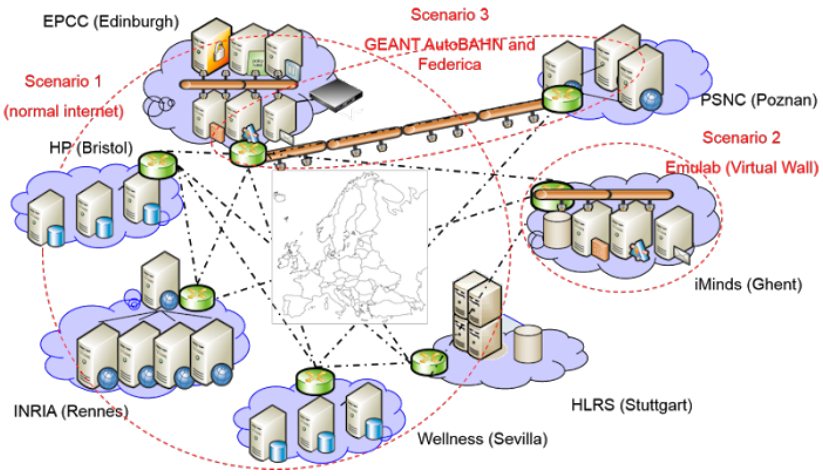


Figure 2.3: BonFIRE testbeds.

to provide important information for software applications in the virtual machines. This testbed also offers elasticity resources, that are dynamically created, updated and destroyed according to execution environment.

Name	CPU cores	Memory	Features
Lite	0.5	256 MB	
Small	1	1 GB	
Medium	2	2 GB	
Large	2	4 GB	
Large+	2	4 GB	Higher CPU clock speed
Large-en	4	4 GB	
Xlarge	4	8 GB	
Xlarge+	4	8 GB	Higher CPU clock speed
Custom	User defined	User defined	VCPU must be an integer

Table 2.1: Instance types of BonFIRE

2.4.2 Federated Tools

In the breast of *Fed4FIRE* project, some tools for deploying, controlling and monitoring the experiments have been developed. Some of these tools are used only for developers to deploy, to provision, to make reservations and to discover resources. These tools are *Flack, Omni and SFI* which they have a common interface named *SFA*.

The experimenters utilize the tools for controlling the experiments. These tools are described as follows:

- *NEPI:*The Network Experimentation Programming Interface, is a life-cycle management tool for network experiments. It is developed in Python and it provides a high-level interface to describe experiments, to provision resources, to control experiments and to collect all the results of the experiment. To highlight that this implementation provides an important way to manage and to make a workflow for an experiment.
- *Object Monitoring Framework v6 (*OMF6*):* is a generic framework that allows the definition and orchestration of the experiments. It can be used for control the experiment through a distributed infrastructure based on Extensible Messaging and Presence Protocol (XMPP) and an Aggregate Manager that manages all the information flows. It can be integrated with Object Monitoring Language (OML) for provisioning, to control and to collect all information about the experiment.
- OML*:* this tool provides a powerful way to take input from any sensor or device with a software interface. This tool also defines and implements a reporting protocol and a collection server. On the client side, any application can be implemented using the OML Application Programming Interface (API) for collecting any information about

the status of the devices concerning the experiment.

2.5 Graphical User Interfaces

A Graphical User Interface (GUI) is a software that facilitates the human-machine interaction visually using images and graphics items. Normally the interactions are produced by user actions directly and these actions corresponds to events. The GUI events are normally produced by the mouse, touchpad, keyboard or nowadays,a touchscreen. When an event is located, a determinate action is performed changing the interface itself, or creating, updating or deleting data or to accomplish a specific proceeding.

2.5.1 Frameworks

There are lots of graphical user interface engines or frameworks available for Python. As the project is developed in Python, the user interface has to be developed in Python also. In addition, the searched engines have to be multiplatform. Finally, considering these above criteria, some frameworks and libraries as *PyGUI, PyGtk, PyGame, PyQt, PyKDE and WxPython* have been studied:

- *PyGUI:* It is a API that provides to make graphical user interface easily, lightweight development and it can be used in any platform or operative system. The PyGUI API offers that programmer does not need read any documentation because it is written in Python. It use PyOpenGL libraries also. It is distributed under General Public License (GPL) v3.

- *PyGtk:* It is a library set of Python with which the programmer can develop programs with a graphical user interface easily. It is multiplatform is distributed under Lesser General Public License (LGPL) licence with few restrictions.

- *PyGame:* Is a set of Python modules that allows to create games and multimedia software as graphical user interfaces. Pygame is fully portable and runs on nearly every platform and operative system. This engine evolves funcionality of Simple Direct-Media Layer (SDL) library, OpenGL and provides Blender integration. It is distributed under GPL v3.

- *PyQt:* is a Python binding for Qt development. Qt is a cross-platform and GUI framework for developing graphical user interfaces. Qt is distributed under GPL v3 and LGPL license. It is developed by Nokia and it permits to develop software for mobile, embedded platforms, desktop platforms and any operative system.

- *WxPython:* is a GUI toolkit for Python that allows to create robust,highly functional graphical user interfaces easily and simply. It is implemented in Python and distributed under GPL v3 license.

2.6 Database

Since computer manage the users information, humans had tried to sort these data and to collect from machines effectively. For this purpose, the databases were created. The databases are conceptual models to store the information orderly. Another step forward was the creation of the Database Management System (DBMS). The DBMS is a software to manage the information contained into database to concentrate, to arrange and to provide lots of mechanisms to manage the information contained into database repository.

2.6.1 DBMS

In order to simulate the constellation of satellites for GEO-Cloud project, some free DBMS were studied. The most interesting for the purpose of this project are as follows:

- **MySQL:**Open Source relational database management system developed by *MySQL AB*. It is the most used database system in web applications and second in the world. *MySQL* provides triggers, cursors, sub-selects, stored procedures, a embedded database library and works with distributed systems efficiently among others. This DBMS is multiplatform and it works in any operative system.

- **MonetDB:** Open Source column-oriented database management system developed by *Centrum Wiskunde & Informatica* in Netherlands. It offers high performance on complex queries in large databases combining tables with lots of columns and multi-million rows. This DBMS is performed in data mining, geographic information systems and others like that.

- **PostgreSQL:** *PostgreSQL* is a object-oriented relational distributed database management system developed by *PostgreSQL Global Development Team*. It is distributed under Berkeley Software Distribution (BSD) license. This database use a client-server model based in multi-processing for granting system stability.

MySQL was selected for developing because there are quite documentation and the syntax is similar to Structured Query Language (SQL).

2.7 Distributed Systems

Nowadays, almost of applications need to obtain information from other sources or communicating with another hardware either other computers or devices. As a result, some programming paradigms like client-server, remote procedure call and remote object invocation were born.

- **Client-Server paradigm**
  This architecture is composed by two parts: the first one is the Server component and the second one is the Client. The server is listening request from Client and when a application comes, the server decodes and processed it and sends back the reply to the

Client.

Remote Procedure Call
Remote Procedure Call uses the same principle as Client-Server but in much complex way. Basically in remote procedure call paradigm, a client calls a function as if it were a local function on the client machine but this called function are located in another host that it may use to perform a function.

Remote Object Invocation
This paradigm was born when the object-oriented programming paradigm was developed. It is based on there are distributed objects and these objects not necessary are fixed in a host. When a distributed-object is created, it is bound to a set of machines as slave, so when a client application recovers that object using a distributed-object registry or another one like that, the provided operations by the distributed object can be remotely performed as if it was a operation call in applicant machine.

2.7.1 Middleware

For client-server paradigm there are not any middleware for developing applications. Normally these kind applications are developing using the standard oriented socket libraries. For Remote Procedure Call (RPC) development, there are several implementations as Simple Object Access Protocol (SOAP), Common Object Request Broker Architecture (CORBA), UNIX RPC and Pyro among others.

For Remote Object Invocation paradigm, novel middleware are. The most used are Hadoop and ZeroC Internet Communications Engine (ICE) and its features are as follows:

2.7.1.1 Hadoop

Hadoop is a framework for developing reliable and scalable distributed applications. It is based on the architecture used by *Google* named "MapReduce" and other components like *Google File System*. Some of its components among others are:

- *Hadoop Common:* The common utilities for other Hadoop modules.
- *Hadoop Distributed File System:* Distributed file system that provides high-throughput access.
- *Hadoop YARN:*A framework for job scheduling and resource management.
- *Hadoop MapReduce:* A YARN-based system for parallel processing of large data sets.

It is designed for scaling up thousands of machines. It is maintained and distributed by *Apache Software Foundation* under GPL license.

2.7.1.2 ZeroC ICE

ZeroC ICE, the internet communications engine, is a modern distributed computing platform with support lots languages as C++,Python, Java and Ruby among others. Permits to developers creates new powerful, efficient and simple applications with minimal effort. The main features of this middleware are:

- It is a general-purpose distributed computing platform.
- Uses a modern and flexible specification language.
- Dynamic invocation and dispatch.
- Request Forwarding.
- Asynchronous and synchronous invocation and dispatch.
- One-way, datagram and batched invocation.
- Fully thread-safe.
- Several transport protocols can be used as Tranport Control Protocol (TCP),Secure Sockets Layer (SSL),UDP and also Internet Protocol (IP)v4 and IPv6 are supported.

ICE provides the following services:

- *IceGrid:* A service for large-scale grid computing applications.
- *IceStorm:* A sophisticated event distribution service.
- *Freeze:* Provides object persistence and the possibility to migrate these objects to a database.
- *Glacier2:* A firewall service.
- *IcePatch2:* A software and files distribution service.

2.8 Software Design

In this section, the software engineering knowledge as design patters and software portability required by this project was studied. These areas are described in the following subsections.

2.8.1 Multiplatform source

One of the main objectives of GEO-Cloud consists of the software portability. As this project is an experiment, it has to be checked in any platform with any operative system, so the

The developed source was written in Python. Among the huge variety of Python interpreters, the used version for developing this project source is Python 2.7.

GNU/Linux was the platform in which the development of all project files were carried out. Besides the libraries distributed within Python, *Paramiko* Python library was used. This

library implements Secure SHell (SSH) protocol operations for managing and controlling machines remotely. This library is distributed under GPL v3. http://www.lag.net/paramiko/

2.8.2 Software Development Life Cycles

Several methodologies for software development are. Historically, the software was developed without any methodology, so there were lots of bugs in the code. As a result, programs did not work correctly or they were inefficient or inclusive, they stopped unexpectedly. Then the saviour programmer tried to solve the trouble wasting lot of time. Thus, the software development is very expensive. At present, the software development is guided by life cycles. In this cycles, stages as planning, designing, implementation, testing, documenting, deployment and maintenance are basics for software development. Depending on how these stages are done, several life cycles are. Among others, the most important and more used are:

- *Waterfall model:* strict model in which developers have to follow these stages in order: analisys,design, implementation, testing, deployment and maintenance.
- *Spiral model:* this life cycle combines the waterfall model and rapid prototyping.
- *Iterative and incremental cycle:* it bases on the development of small but ever-larger portions of a software. During software development, several iterations may be in progress at the same time. Using this cycle, the end-users can check the no-ended product for updating requirements or to give feedback to end-user.
- *Agile development:* This model is very used nowadays, and its principles are the iterative development, to incorporate continuous feedback and stages iterations.

2.8.3 Design Patterns

The software programming patterns that were studied for developing GEO-Cloud are:

- Singleton: only a single instance of a class can exist.
- Composite: a tree structure of simple and composite objects.
- Proxy: an object that represents another one.

2.8.4 Testing

At the same time the development of GEO-Cloud software, the testing process was done also. The testing was made by two ways: hand testing and test-case oriented testing. In case of the second one, sundry testing frameworks were studied. The selected framework for testing in Python was *UnitTest* and its extension *Nose*. It is included in Python standard library, is easy to use and it has lots of plugins.

The developed test-cases were black-box essentially. Some white box test-cases were developed but the code coverage was not thorough.

Chapter 3

Objectives

In this chapter the objectives carried out in this project are included.

3.1 General Aim

The GEO-Cloud project main objective can be defined as the modelling and implementation of a close to real world EO System in *Fed4FIRE* to validate the *Fed4FIRE* infrastructure.

3.2 Specific Objectives

The main objective of this project is defined regarding a series of functional requirements as follows:

- **A Cloud Infrastructure in BonFIRE Testbed**: This infrastructure for EO provides the ingestion of raw data, on demand processing the raw data to obtain images, and later storing and cataloguing transparently, dynamically and automatically. For this purpose this components are needed:
  - **Cloud Orchestrator**: It manages the processing of images obtained from satellites and controls all the stages (ingestion of images, processing, storing and cataloguing). The images processors are provided by *Elecnor Deimos* company, so their implementation is proprietary and they are treated as a black box.
  - **Archive and Catalogue Subsystem**: The results obtained from the processing have to be available in the cloud infrastructure for end users.
  - **Processors Module**: It process the raw data sent from the Orchestrator component obtaining processed images from the Earth.
- **A profiling tool in PlanetLab**: for obtaining the real measures of parameters and metrics involved in the communications of all components as clients, ground stations, cloud and satellite constellation.
- **Software in Virtuall Wall**: This software simulates the satellite constellation getting images and downloading into the ground stations, ground stations receiving the raw data sent by the satellites and end users accessing the resources in cloud through web

services. Also, the connections between the ground stations and the cloud infrastructure have a set of impairments. This values of these impairments are obtained with the profiling tool in *PlanetLab* in order to build a realistic experiment.

- **Creation of the experiment**: By combining and integrating the developed in the three testbeds mentioned before.
- **Creation of a Graphical User Interface**: This GUI provides the deploy,execution and stop automatically.
- **System Validation**: Simulation of a scenario to validate the entire system obtaining, downloading, processing, storing and cataloguing EO images. Also, to validate if future internet cloud computing and networks provide viable solutions for conventional EO Systems to establish the basis for the implementation of EO infrastructures in cloud.

Chapter 4

Method of work

In this chapter, the development software methodology during the development of GeoCloud is described. Also, the tools such hardware and software used for is depicted.

4.1 Development Methodology

The selected methodology has been the *iterative and incremental* model. This model provides numerous advantages as early adaptability, testing continuously and to have a implementation quite close to the final product. A scheme of the model is shown in figure 4.1.

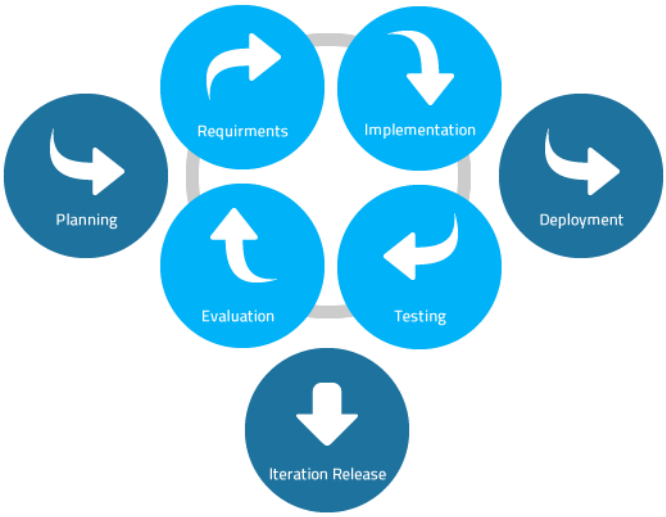


Figure 4.1: Iterative-incremental model

This methodology enforces a strategy based in small iterations in which ones there are short changes applies to each module. This permits to generate a prototype growing and closing to the final version validating the initial requirements. In this way if any changes are required by the advisor, is easily to carry out them.

As the project has several parts, it is divided in modules and simultaneously, each module is divided in iterations. These iterations are depicted in Section 6.1.

4.2 Tools

In this section the resources such as software and hardware used during the development are enumerated and described.

4.2.1 Programming Languages

Several programming languages have been used for the Geo-Cloud implementation. These languages are described in Section 2.2 and they can be summarized as follows:

- **Python:** Main language used in the project. It is a multiplatform, multipurpose object oriented language, although permits imperative and functional programming. It does not need to compile the source because it is interpreted (the operative systems in the cloud maybe not the same).
- **XML:** is a mark-up language that defines a set of rules for encoding documents in a format. It is used in some configuration files used by the Orchestrator module or the User Interface tool.
- **JSON:** is a lightweight language for data interchange between applications. The *BonFIRE* experiment descriptor must be written in this language indicating the virtual machines instances, physical machines and the network resources instantiated.
- **Bash:** consists of several sentences that the operative system is able to read and translate in order to play a specific action. All the currents *Linux* distributions contains a Bash interpret.

4.2.2 Hardware

The development of GEO-Cloud project has been carried out in a PC owned by Deimos with the following features:

- *Intel Core i5 3450 3.1 GHz*
- *8 GB RAM*

For the Version Control, the chosen server is Bitbucket [1].

The used resources provided by the *Fed4FIRE* testbeds are the following:

- **PlanetLab**: It currently provides 1204 nodes at 593 sites around the world. The selected nodes for carrying out this project are shown in Table B.1 and Table B.2.
- **Virtual Wall**: It consists of 100 nodes (dual processor o dual core servers) interconnected via a non-blocking 1.5 Tb/s Ethernet switch. 29 nodes have been used to simulate the satellite constellation and ground stations.
- **BonFIRE**: Nodes from diferent testbeds have been selected. The features of each node are depicted in Table 2.1

[1] http:www.bitbucket.org

- From INRIA: 2 nodes Medium.
- From EPCC: 1 node Xlarge for Orchestrator and a Elasticity as a Service (EAAS) for Processing Chain with at least 1 and maximum 60 active Xlarge resources. «MODIFICABLE»
- From IBBT: 1 Shared Storage.

4.2.3 Software

Then, the tools and libraries used for the project are depicted:

- **Operative Systems**
  - *Ubuntu*: UNIX based operative system in which the development has done. It is based in Debian and distributed under GPL v3.
  - *Debian*: UNIX based operative system distributed under GPL v3. It provides more than 37500 precompiled software packets which can be installed easily. The *BonFIRE* , *Virtual Wall* and *PlanetLab* machines run this operative system.
- **Software Development Tools**
  - *Emacs*: versatile and powerful text editor developed by Richard Stallman. This tool is used with *Pymacs* together. Version used 23.2.
  - *JFed*: tool developed by *IMinds* (University of Gent)[2] for deploying nodes from *Fed4FIRE* testbeds.
  - *BonFIRE Web Interface*: *BonFIRE* tool that provides the experiment deployment manually or to upload an experiment descriptor for automatic deployment.
  - *Ipython:* command shell for interactive computing in Python. It allows to check some clauses before adding it into a software. For agile development, it is a useful tool.
- **Graphics and documentation**
  - *LaTeX*:is a document mark-up language widely used for the communication and publication of scientific documents. It is multiplatform and is distributed under GPL v3.
  - *GIMP*: is the *GNU Image Manipulation Program*. It allows some achievements as photo retouching, image composition and image authoring among others. It is multiplatform and is distributed under GPL v3.
  - *Dia*: multiplatform software for drawing many types of schemes. Some design schemes of this document were performed on it.

[2]http:www.jfed.iminds.be

  - *Libre Office:* free open source office suite, developed by *The Document Foundation*. This suite provides such software as *Draw* and *Writer* among other programs.

- **Software Libraries**

  - *Python-mysql*: *Python* library that provides a high-level interface *MySQL* programming.
  - *Python-mathplotlib*: *Python* library that provides some tools and functions for plotting and representing data.
  - *Python-xml*:*Python* library used to manage XML files.
  - *Paramiko*: *Python* library that provides high-level interfaces for connecting through SSH to other network host.
  - *PyQt:* Python binding for the *Qt* cross-platform framework. *PyQt* is available in two versions, *PyQt4* and *PyQt5*. *PyQt5* was used for developing the project.
  - *Phonon:* multimedia API provided by Qt for handling multimedia streams. For developing the GUI of this project, this library was used.

Chapter 5

The Geo-Cloud Experiment

The GEO-Cloud experiment consists of the emulation of a realistic and complete Earth Observation System that provides services using cloud technology. For that purpose a constellation of satellites, ground stations, a cloud architecture, use cases and end users' models are designed.

The EO system will be computed and emulated in *Fed4FIRE*: A constellation of satellites record images of the Earth in a daily basis. The images are transferred to ground stations that ingest the data into a cloud computing infrastructure. The data is processed and distributed to end users.

The GEO-Cloud experiment is tested in three testbeds: *Virtual Wall* ,*BonFIRE* and *PlanetLab* . GEO-Cloud is divided into two sub-experiments:

- One experiment in a system integrated in *Virtual Wall* and *BonFIRE* .
- One experiment in *PlanetLab* .

The experiment in *Virtual Wall* and *BonFIRE* emulates the whole EO system. The system is constituted by:

- A *Space System Simulator* including models of satellite constellation and a ground stations network.
- A cloud computing system that implements a novel architecture that integrates an *Orchestrator*, , an *Processing Chain* component based onEAAS with image processors to transform the raw data acquired by the satellites and transform it into orthorectified images, and finally an *Archive and Catalogue* module for storing, cataloging and allocating images.

The experiment in *PlanetLab* consists of the emulation of the networks that constitute the links between the ground stations to the cloud and from the cloud to the end users. There, the network performance, features, bandwidth and impairments are monitored and measured. Those parameters once measured are used to update the models implemented in *Virtual Wall* , i.e, bandwidth, latency and loss-rate.

In the next sections, the detailed design of the GEO-Cloud experiment is fully described.

5.1 Satellite System Design

5.1.1 Design of the flight and ground segments

In this subsection, the main characteristics of the system are presented. Objectives, constraints, previous estimations and possible modifications and their effects in the system are exposed.

In a wide vision, it is an EO system consisting of a constellation of satellites equally spaced in a Low Earth Orbit (LEO) orbit with the aim of achieving daily coverage of the entire Earth surface. These conditions imply a very sophisticated handle of a huge quantity of data.

The following requirements have been fulfilled to design the system:

- Swath: $160km$ (based on state of the art cameras).
- Resolution: $6.7m$ (based on state of the art cameras).
- Low Earth Orbits.
- Sun Synchronous orbits.
- Download data rate: $160Mbps$ (based on Deimos-2 satellite characteristics).
- Optical Bands: 5 Multispectral (based on state of the art cameras).

5.1.1.0.1 Global Daily Coverage

The objective of this system is the acquisition of images of the total Earth surface in a daily basis. Global coverage is considered to include the land surface that is shown in Figure 5.1.

Figure 5.1: Land surface to be acquired in a daily basis

5.1.1.0.2 Flight Segment

The satellites are selected according to the current state of the art. However, some enhancements can be assumed as a way to adjust the analysis to the short term future.

5.1.1.0.2.1 Satellite performances

As a first iteration for the system, small platforms of less of $500kg$ are supposed according to the size and payload bay characteristics of representative platforms in this category. This study is based on the bus platforms currently offered by Surrey Satellite Technology Ltd, Satrec Initiative, Sierra Nevada Corporation, etcetera.

To reduce the amount of satellites orbiting the Earth, a design with two payloads has been assumed. Thus, the swath (the width of the field of view of each satellite in the surface of the Earth) can be duplicated without decreasing the resolution.

The satellites shall be pointing to nadir, acquiring images without maneuvering because of the acquisition plan for cover the Earth daily. Simultaneously, when a satellite gets into the field of view of a Ground Station, the download starts and at the same time the satellite continues imaging.

The main specifications of the satellites in the constellation are shown in Table 5.1. The values of each parameter have been selected according to the state of the art and the desired quality of the images in the mission.

Specification	Value
GSD	$6.7m$
Swath	$160Km$
Number of bands	5
Digitalization	$12bits$
Download Data Rate	$160Mbps$
Compression Rate	$2:1$

Table 5.1: Main Performances of the Satellites

5.1.1.0.2.2 Orbit definition

It is common in EO satellite missions the use of *sun-synchronous* orbits. These orbits guarantee that the lighting conditions of the imaged places are the same during the mission, which is a very desirable characteristic.

Local Time Ascending Node (LTAN) is also a desired condition of the orbit very related to the lighting and weather conditions of those places that the satellite overflies (LTAN is selected according to the desired local time of the overflown places and the cloud formation during the day); it is common the use of *LTAN 10:30h* for Earth Observation.

Other of the main parameters of an orbit is the altitude. Altitude has effects in the resolution and the swath of the satellites, which has impact in the number of satellites required to achieve the coverage objective. According to the value of those parameters in the payloads included in the satellites, the reference altitude for the system was found to be *646km*. Sun

Synchronous Orbit (SSO) condition implies a relation between the altitude and the inclination of the orbit of *97.97deg* in this case.

5.1.1.0.2.3 Number of satellites in the constellation

The number of satellites shall be calculated using the altitude ($646km$), the inclination ($97.97deg$) and a swath in Table 5.1. As a result, 17 satellites are required to carry out this mission. In Figure 5.2 the whole constellation is shown.

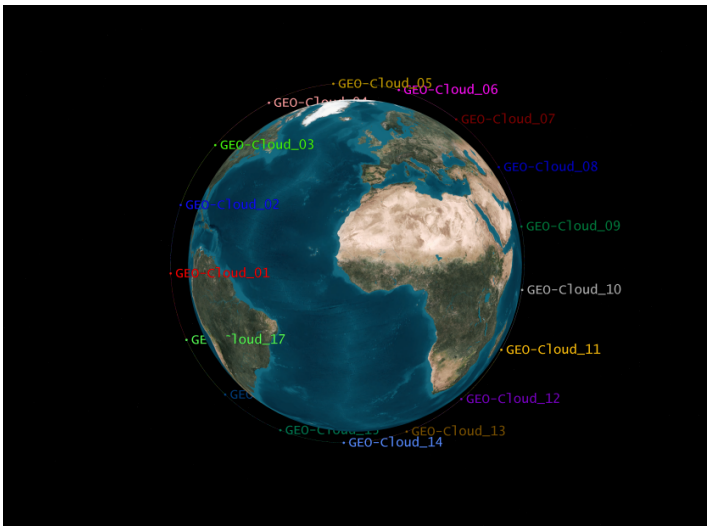


Figure 5.2: Constellation of 17 satellites in a SSO orbit at 646km

5.1.1.0.3 Ground Stations design

When the satellite acquires the data, it has to be downloaded to the Ground Stations and then it has to be distributed to the customers. Due to the huge quantity of data and the limitations the download data rate sets, several stations distributed over the surface of the Earth are required. They will allow the satellite to communicate with them and download the images. Due the accumulated duration of the accesses to all the ground stations per day and the frequency of accesses per day (it varies with the latitude of the station), the Figure 5.3 shows how the Ground Stations and its footprints (area in which the satellites can communicate with the Ground Station) is distributed.

5.1.2 Generated Data Volume

Specifically, $135,698,500km^2$ of ground surface are daily acquired. With the following expression can estimate the data volume generated:

$$Acquired\ Data\ Volume = \frac{Acquired\ Surface}{GSD^2} * N^o bands * Digitalization \qquad (5.1)$$

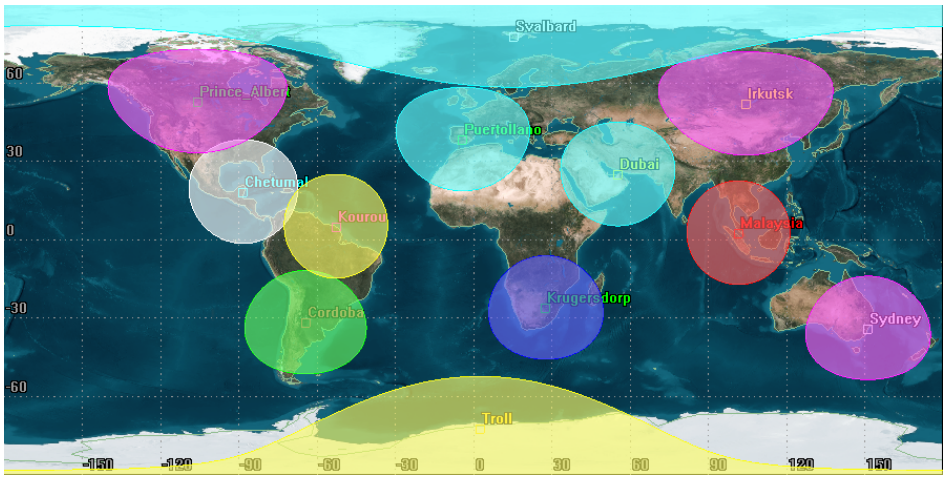


Figure 5.3: Footprints of the selected Ground Stations

Before downloading the images they are compressed. The ancillary data is included in the process (auxiliary information useful for the geolocation of the images, protocol . . .). In this case, the ancillary data is estimated to be 12% of the acquired data (based on Deimos 2 satellite measurements), which is added and then compressed. With the values depicted in Table 5.1, the ancillary and the daily ground surface adquired, the data on ground can be estimated. As result including ancillary data before compression, *11.55TBytes* shall be downloaded daily.

5.2 Satellite System Development

This section presents the *Space System Simulator* that emulates the real behaviour of a satellite constellation of 17 satellites that download images to a network of 12 ground stations connected with the *BonFIRE* cloud. The simulator is implemented in *Virtual Wall* .

The *Space System Simulator* is constituted of three components:

- *Satellite System Simulator* : It simulates the dynamics and communications of the constellation of 17 Earth observation satellites.
- *Ground Station System Simulator* : It simulates the dynamics and communications of the network of 12 ground stations distributed around the World.
- *Distributed Database:* It contains all the required information and parameters to initialize the simulators and make them run in every specific simulator. While the *Satellite System Simulator* and the *Ground Station System Simulator* are implemented in *Virtual Wall* , the distributed database is computed in the *BonFIRE* cloud to allow the access of the two simulators.

To adapt the performance of the *Space System Simulator* to the Fed4FIRE testbeds, some located scenarios were designed in order to reduce the amount of data to process, store and

distribute during the simulations. Thus the simulation is shortened to a specific time required to acquire and download certain areas of interest *(AOI)*.

The detailed design of the *Space System Simulator* and its implementation in *Virtual Wall* are presented.

5.2.1 Image Acquisition

The first step is to implement the acquisition of images by the satellites of the six predefined scenarios «(for an extended description on the scenarios see GEO-Cloud-D10.8-Detailed design report-2014-01-31)» with the satellite constellation:

1. Emergencies – Lorca Earthquake (Spain)
2. Infrastructure monitoring. Affection in railway infrastructures by sand movement in desert areas (Spain)
3. Land Management – South West of England
4. Precision Agriculture – Argentina
5. Basemaps – Worldwide
6. Online Catalogue / Ordering – Worldwide

In each scenario, an Area of Interest *(AOI)* is defined for the satellites to acquire it during the simulation. In orbit satellites are nadir pointing in order to acquire images of the sub satellite point over the Earth surface as shown in Figure 5.4.

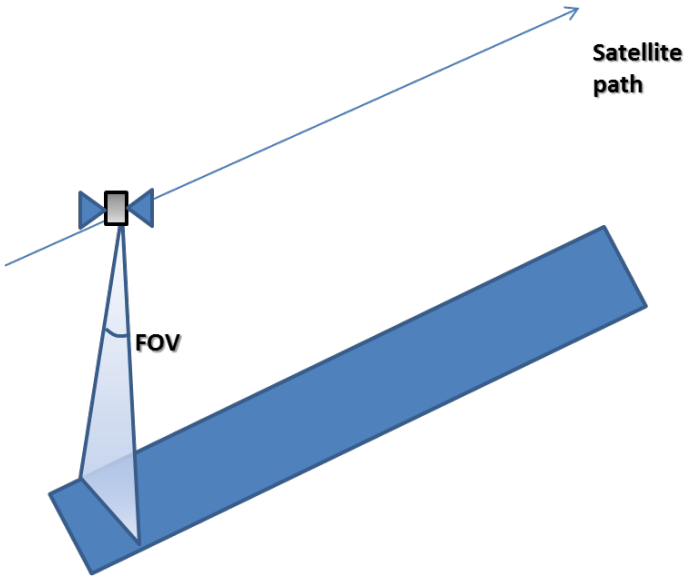


Figure 5.4: Example of strip imaging

In the next subsection the assumptions that were made to simulate the system as realistically as possible are described for the scenarios. Note that scenarios 5 and 6 have the same

AOI, which is the whole land mass on Earth. This involves that because of the huge amount of data that has to be recorded, specific assumptions to the scenarios 5 and 6 were made according to the Fed4FIRE testbeds' limitations.

5.2.1.1 Assumptions in satellite image acquisition

The *AOI* in each scenario can be acquired by one or more satellites depending on the size of the *AOI* relatively to the scene size (note that the GEO-Cloud satellites have a swath of $160km$, thus we divide the acquisition into scenes of $160kmx160km$). In each scenario we call *main satellites* to the satellites with the task of acquiring the *AOI* (note that all satellites are *main satellites* in the model for the scenarios 5 and 6).

Along the duration of the scenario other satellites acquire images of the areas of the Earth surface they are passing over. Those images are not in the area of interest.

During the experiments in *BonFIRE* , they will be processed but not stored into the system, since the focus of the mission in every scenario has to be in the defined area of interest. Those non *AOI* images allow us to emulate a real system, since we take into account all the possible inputs to the system (note that in the scenarios 5 and 6 all the images created are *AOI*).

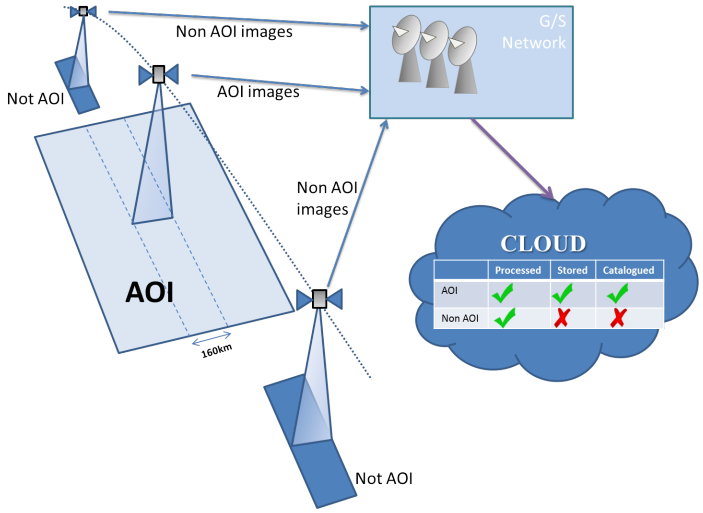


Figure 5.5: Diagram of images adquisition

Thus, those Non *AOI* images, during the simulation will be acquired, downloaded to the ground stations, transferred to the cloud and processed, but not stored, neither catalogued. In addition, it has to be taken into account that the main satellites can also acquire some images out of the *AOI* during the duration of each scenario. Those acquired images are also considered Non *AOI* images.

In every scenario the time is set to 0 when the simulation starts in the Fed4FIRE environment.

These assumptions are summarized as follows:

1. Only the scenes acquired by a satellite that include the *AOI* are considered to carry out the complete simulation.
2. Images taken by the satellites that are out of the *AOI* are processed but not stored into the system to adjust the experiment execution to the resources provided by *BonFIRE* .

5.2.1.2 Types of acquisition of the AOI by a single satellite

Depending on the relative sizes of both the *AOI* and the scenes, three different situations can occur when a single satellite is acquiring images:

1. Simple acquisition: the *AOI* (at least the part to be imaged by the satellite) fits into just one scene.

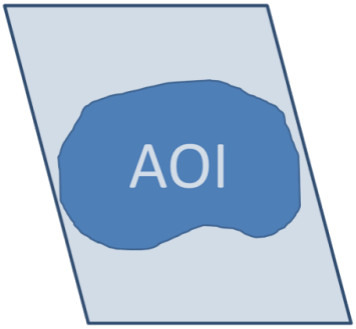


Figure 5.6: Simple acquisition

2. Multiple consecutive acquisitions: the *AOI* to be acquired by a satellite fits in a strip with several scenes.

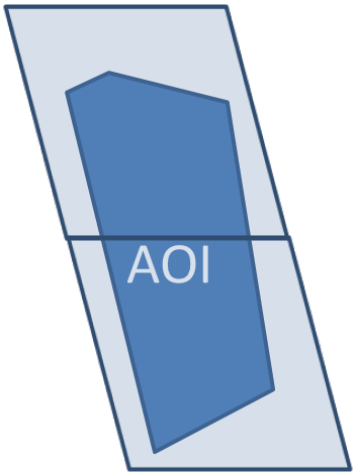


Figure 5.7: Multiple consecutive acquisitions

3. Multiple non-consecutive acquisitions: the *AOI* to be acquired by a satellite fits in a strip but there are some scenes between the acquisitions that have to be acquired in

order to complete the general mission (world map daily) but it is not necessary for the scenario.

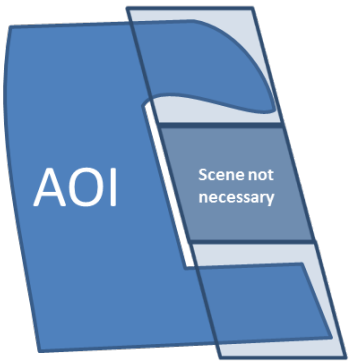


Figure 5.8: Multiple non-consecutive acquisitions

5.2.1.3 Parameters required for the simulation of the scenarios

For each scenario, the following parameters and assumptions are considered:

1. *S: Scenario number.* It indicates the scenario that is being simulated (from 1 to 6).
2. *Main satellites:* The main satellites are those that acquire images of the AOI in each scenario. Notice that all the satellites are numbered from 1 to 17.
3. *ADR:* Acquisition Data Rate. Rate at which the images are acquired by the satellite.

$$ADR = 1395Mbps \tag{5.2}$$

4. *CR:* Compression Rate. Rate at which the images are compressed in the satellite for their download.

$$CR = 14.1 \tag{5.3}$$

5. $Bandwidth_{sat}$*:* Download rate between the satellites and the ground stations.

$$Bandwidth_{sat} = 160Mbps \tag{5.4}$$

The previous bandwidth is used as follows:

- The satellite is downloading images in memory and at the same time is acquiring images and downloading them:

$$Bandwidth_{sat} = Bandwidth_{(sim\_acq)} + Bandwidth_{mem} \tag{5.5}$$

where $Bandwidth_{(simacq)}$ is the bandwidth used for downloading images that are being acquiring at the same time and $Bandwidth_{mem}$ is the bandwidth used to

download images in the satellite memory. $Bandwidth_{(sim\_acq)}$ can be obtained as follows:

$$Bandwidth_{(sim\_acq)} = ADR/CR = 98.9Mbps \tag{5.6}$$

where *ADR* is the Acquisition Data Rate (Mbps) and *CR* the data compression rate. And $Bandwidth_{mem}$ can be obtained as follows:

$$Bandwidth_{mem} = Bandwidth_{sat} - Bandwidth_{(sim\_acq)} = 61.1Mbps \tag{5.7}$$

6. T_0*:* Start time. It is the start time of the scenario. The scenario starts when the first main satellite begins the acquisition of the first portion of *AOI*.
7. T_f*:* End time. It is the time when the last main satellite finishes the downloading of the *AOI* taken images.
8. $[T_{AOI}]_{0i}$*:* Time a main satellite starts acquiring the *AOI*. i stands for the number of the main satellite.
9. $[T_{AOI}]_{fi}$*:* This is the time a main satellite finishes acquiring a piece of *AOI*.
10. $[\Delta T_{AOI}]_i$*:* This is the difference $[T_{AOI}]_{fi} - [T_{AOI}]_{0i}$. Note: Some satellites (all of them in the model for scenarios 5) and 6) can require more than one orbit to acquire the corresponding *AOI*, and then these satellites will have a different $[\Delta T_{AOI}]_i$ in each acquisition of *AOI*. $[\Delta T_{AOI}]_i$ is a multiple of $T_{scene} = 23.4s$, which is the needed time to acquire one scene of $160kmx160km$, considering that satellite velocity on ground is $v_{Gsat} = 6.84km/s$:

    $$T_{scene} = L_{scene}/v_{Gsat} \tag{5.8}$$

    $[\Delta T_{AOI}]_i$ can be calculated as follows:

    $$[\Delta T_{AOI}]_i = nL_{scene}/v_{Gsat} = nT_{scene} = [T_{AOI}]_{fi} - [T_{AOI}]_{0i} \tag{5.9}$$

    where n is the number of scenes, L_{scene} is the length of the scene (in this case $160km$), and v_{Gsat} the velocity of the satellite on ground (in this case $6.84km/s$).
11. $[T_{GS}]_{0ij}$*:* Time a main satellite i enters into the visibility cone of a *Ground Station* j.
12. $[T_{GS}]_{fij}$*:* Time a main satellite i leaves the visibility cone of a Ground Station j.
13. $[\Delta T_{GS}]_{ij}$*:* This is the difference $[T_{GS}]_{fij} - [T_{GS}]_{0ij}$.
14. T_{start}*:* This is the start time of the simulation. We chose to be 5 seconds in order to synchronize all the satellites.

As an example, the Table 5.2 shows the data was obtained from "Scenario 2: Infrastructure monitoring affection in railway infrastructures by sand movement in desert areas":

5.2.2 Image Downloading

The acquired images by the satellites shall be downloaded to the emulated ground stations through the antennas network designed «(see GEO-Cloud-D10.8-Detailed design report-2014-01-31)». All the scenarios finish at T_f, when the last main satellite completely downloads the last acquired portion of the *AOI*. During the simulation, the rest of the satellites not overflying the *AOI* would be imaging other places; these images will also be downloaded and processed in parallel to the *AOI* images to simulate a realistic case, but as previously explained those images will not be stored nor catalogued.

Because of the limitations in the testbeds (limited storage and compute resources in *BonFIRE* cloud and multiplexed channel over virtual networks in *Virtual Wall*) it is necessary to scale the data involved in these simulations from the original design of the GEO-Cloud experiment. It has been decided to change the designed compression rate from lossless compression $2:1$ to a rate compression $14:1$. With this change, the data volume has been reduced and the size of every $160kmx160km$ scene after compression is now $288MB$ instead of the almost $2GB$ size of the original images.

In order to download the data, the satellites have to be inside the visibility cone of a ground station. During these accesses, the satellites download images at a rate of $160Mbps$. Part of these images are in memory before entering the visibility cone and others are being acquired simultaneously during the downloading task (see section 5.2.1.3 for a wider explanation of the download parameters and Figure 5.9).

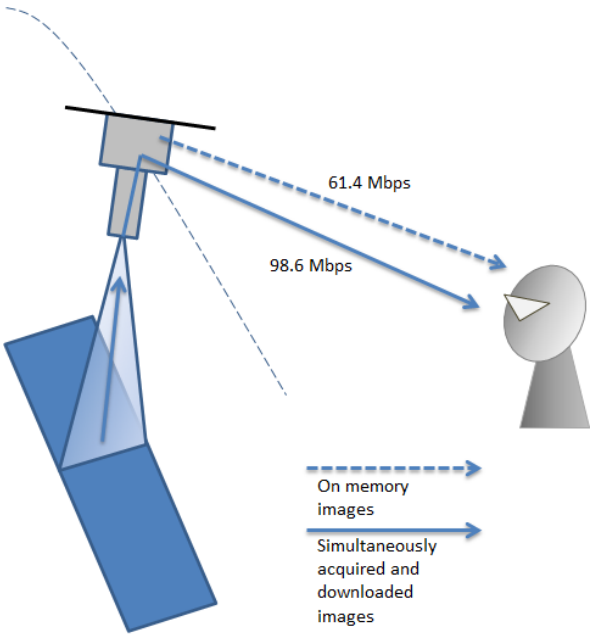


Figure 5.9: Multiplexed images downloading

It is possible that the duration of the accesses and the time to download an image are not multiples of $T_{scene} = 23.4s$. In this case, an image would be partially downloaded at the

end of the access but it would be supposed that this image is completely downloaded in the following access in other ground station.

For the memory management, a *LIFO* (last in, first out) procedure is applied. This implies that the latest acquired images will be downloaded as soon as possible. This imposes that when a satellite is inside the visibility cone of a ground station both the images latest acquired and the images that are being acquired of the *AOI* will be simultaneously downloaded by multiplexing them. Note that because of differences between the rate of acquisition and downloading, and also because of the compression process, the downloading of images that are being simultaneously acquired do not occupy all the bandwidth of $160Mbps$; the portion not occupied is used to download on board storage also with a *LIFO* process (See 2.3 numbers 3 to 5).

As an example, the data of the accesses for the Scenario 2 are included in the following table:

Next, a summary of the assumptions made for the image downloading from the satellites to the ground stations are described:

1. Time to process the images on board before downloading is considered to be instantaneous. Then, the simultaneous acquisition and download of the *AOI* inside the visibility cone is done without latency.

2. The ground stations network was designed to guarantee the download of a complete world map in a daily basis. With this assumption into account, the management of the memory on board of each satellite does not require an additional design. This involves that we do not know in which ground station all the areas acquired out of a visibility cone are downloaded.

3. Because of the small gap between the satellites, which are in the same orbit, more than one antenna in each ground station is required to be available. This provides the possibility of having simultaneous contacts with all the satellites inside the visibility cone.

5.2.3 Getting the satellite data

The steps to obtain and simulate the realistic behaviour of the satellites are next described.

5.2.3.1 Extraction of the real behaviour of the satellite constellation in the scenarios

The first point is to obtain the realistic behaviour of the constellation of satellites designed in «"GEO-Cloud-D10.8-Detailed design report-2014-01-31.docx"». For that purpose, the constellation of satellites and the ground stations were implemented in the $SystemsToolKit^{®}$

Scenario	Start Time	End Time	Main Satellites	Start Acquisition	End Acquisition	Number of *AOI* scenes
2	54060	54753.4	4	54060	54083.4	1
			3	54390	54413.4	
			2	54730	54753.4	

Table 5.2: Example of data of image acquisition for Scenario 2

Sat. Ident. Number	Ground Station	Access Start Time (s)	Access Stop Time (s)	Duration of the passes (s)
1	Krugersdorp	54184.2	54516	331.8
2	Dubai	54184.2	54516	331.8
2	Krugersdorp	54549.4	54753.4	204
3	Dubai	54060	54753.4	204
4	Dubai	54060	54337.5	277.5
4	Svalvard	54660	54753.4	93.4
5	Svalvard	54316.3	54753.4	437.1
6	Prince Albert	54708.1	54753.4	45.3
6	Svalvard	54060	54639.4	579.4
7	Prince Albert	54362	54753.4	391.4
7	Svalvard	54060	54296.7	236.7
8	Prince Albert	54060	54577.9	517.9
9	Prince Albert	54060	54246.8	186.8
15	Troll	54419	54670.3	251.3
16	Troll	54085.6	54303.4	217.8
17	Krugersdorp	54495.4	54753.4	258

Table 5.3: Example of data of accesses for Scenario 2

(STK) software.

The next steps were followed to obtain the values of the parameters required for the simulations of the defined scenarios:

- Commonly for all the scenarios:
  1. Implementation of the satellite constellation.
  2. Implementation of the ground station network.
  3. Calculation of the access time for each satellite communicating with each ground station during 24 hours.
- For each different scenario:
  4. Identification of the main satellites.
  5. Extraction of the times each satellite is acquiring the AOI: $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$.
  6. Calculation of the number of scenes acquired: n.
  7. Extraction of the access duration of each satellite with the ground stations it communicates within the scenario duration: $[T_{GS}]_{0ij}$ and $[T_{GS}]_{fij}$.
  8. Start time of the scenario: T_0.
  9. End time of the scenario: T_f.

5.2.3.2 Exportation of data to the simulator

Once obtained the previous data, it is exported to different "CSV" format files:

- The *Scenario_<NUM>_<SCENE>.csv* files: it contains the information of the $[T_{GS}]_{0ij}$ and $[T_{GS}]_{fij}$ for each scenario. *<NUM>* indicates the number of the scenario and *<SCENE>* the name of the scenario. In the GEO-Cloud experiment we simulate 6 scenarios. The previous parameters are defined in the following table:

  For example, the format of the resulting *Scenario_1_ Emergencies_Lorca_Earthquake.csv* file with the $[T_{GS}]_{0ij}$ and $[T_{GS}]_{fij}$ information for the Lorca scenario, looks like the code extract shown in Listing 5.1.

```
"GEO-Cloud_005-To-Troll - Access","Start Time (EpSec)","Start Time (
   UTCG)","Stop Time (EpSec)","Stop Time (UTCG)","Duration (sec)"
1,63638.000,11 Aug 2014 10:10:38.000,63661.400,11 Aug 2014
   10:11:01.400,23.400

"GEO-Cloud_006-To-Troll - Access","Start Time (EpSec)","Start Time (
   UTCG)","Stop Time (EpSec)","Stop Time (UTCG)","Duration (sec)"
7,63638.000,11 Aug 2014 10:10:38.000,63661.400,11 Aug 2014 10:11:01
```

Listing 5.1: Extract of the *Scenario_1_Emergencies_Lorca_Earthquake.csv* of the Lorca scenario

In the code, a heading is separated into columns. It is divided as the table 5.4 shows.

- The *All_Scenarios.csv* file: it contains the information of the $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$ for all the scenarios. A piece of the *All_Scenarios.csv* file that contains the $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$ of all the scenarios is shown in Listing 5.2. It also contains T_0 ,T_f and the number n of *AOI* obtained images.

```
Scenario,Start Sce,End Sce,Sat,Start sat,End Sat,Images
1,63638,63661.4,11,63638,63661.4,1
,,,,,,
2,54060,54753.4,4,54060,54083.4,1
,,,3,54390,54413.4,1
,,,2,54730,54753.4,1
,,,,,,
3,62480,63865.4,15,62480,62503.4,1
,,,14,62821,62844.4,1
,,,13,63161,63184.4,1
,,,12,63501,63524.4,1
,,,11,63842,63865.4,1
```

Listing 5.2: Extract of the *All_Scenarios.csv* code of the Lorca scenario

This file was manually computed. The required fields to describe the behaviour of the satellites acquiring the *AOI* are the depicted in Table 5.5.

Column Title	Function
GEO-Cloud_sat-To-station	Represents the access of a satellite to the specific ground station. sat represents the satellite number and station the ground station name.
Start Time (EpSec)	Means $[T_{GS}]_{0ij}$.
Start Time (UTCG)	Means $[T_{GS}]_{0ij}$ in UTCG format
Stop Time (EpSec)	Means $[T_{GS}]_{fij}$
Stop Time (UTCG)	Means $[T_{GS}]_{fij}$ in UTCG format.
Duration (sec)	Means $[\Delta T_{GS}]_{ij}$.

Table 5.4: Columns headings *Scenario_<NUM>_<SCENE>.csv* files

5.2.3.3 Processing of the Scenario_<NUM>_<SCENE>.csv and All_Scenarios.csv

A script is designed and developed to get and merge the data from the *Scenario_<NUM>_-<SCENE>.csv* and the *All_Scenarios.csv* files, and to store the information in the database:

setDatabase.py. It was developed in Python 2.7 but it is also compatible with all Python versions.

This has to be done because as previously explained *Scenario_<NUM>_<SCENE>.csv* contains the $[T_{GS}]_{0ij}$ and $[T_{GS}]_{fij}$ parameters of each single scenario and *All_Scenarios.csv* the $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$ of all the scenarios. Thus the script matches the $[T_{GS}]_{0ij}$ and $[T_{GS}]_{fij}$ of one scenario with the $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$ of such a scenario.

To execute *setDatabase.py* the arguments depicted in Table 5.6 are required.

An example of the execution of the programme is the following:

> python setDatabase.py 192.168.0.2 All_scenarios_file.csv Scenario_1_Example1.csv Scenario_2_Example2.csv Scenario_3_Example3.csv

5.2.4 Space System Simulator

The *Space System Simulator* is constituted by the following modules:

- The Database
- The *Satellite System Simulator*
- The *Ground Station System Simulator*

To simulate every scenario, on the one hand, the *Satellite System Simulator* is executed. It is constituted by 17 *Satellite Simulators* of individual satellites; each one reproduces the characteristic behaviour of every single satellite in the constellation. On the other hand the *Ground Station System Simulator* is also executed. As in the case of the *Satellite System Simulator* , the *Ground Station System Simulator* is constituted of 12 *Ground Station Simulators* of individual ground stations reproducing the behaviour of every single ground station of the designed network.

The diagram in Figure 5.10 depicts a scheme representing the *Space System Simulator* .

The Space System Simulator sequentially follows the next steps during the execution:

- First, *setDatabase.py* is executed. The script fills the database fields with the information of every scenario.
- Second, the *Ground Station System Simulator* is executed. The *Ground Station System Simulator* can execute all the *Ground Station Simulators* or some of them individually to simulate faults in the ground stations as it can occur in reality. Thus, when a satellite needs to download data into a ground station that is offline, it will receive an exception so the downloading of images does not start.
- Third, the *Satellite System Simulator* is executed. As in the previous case, the *Satellite System Simulator* can execute all the *Satellites Simulators* or some of them individually for the same reason.

Column Title	Function
Scenario	Scenario number. Indicates the scenario executed
Start Sce	Indicates the start of the scenario. It is T_0
End Scen	Indicates when the scenario finishes. It is T_f
Sat	Indicates the number of the *main satellite*.
Star Sat	The time when the satellite starts acquiring the *AOI*. It is $[T_{AOI}]_{0i}$.
End Sat	The time when the satellite finishes acquiring the *AOI*. It is $[T_{AOI}]_{fi}$
Images	This is the number of *AOI* images. It is n.

Table 5.5: Columns headings of the *All_Scenarios.csv* file

Argument Position	Meaning
1	IP address of the host which contains the MySQL database
2	The relative path or absolute path of the file that contains the information of all scenarios (see Listing 5.2)
3...	The *Scenario_-<NUM>_-<SCENE>.csv* files that contain the events occurred in a particular scenario. At least one file must be introduced, otherwise an execution error is produced.

Table 5.6: Arguments of *setDatabase.py*

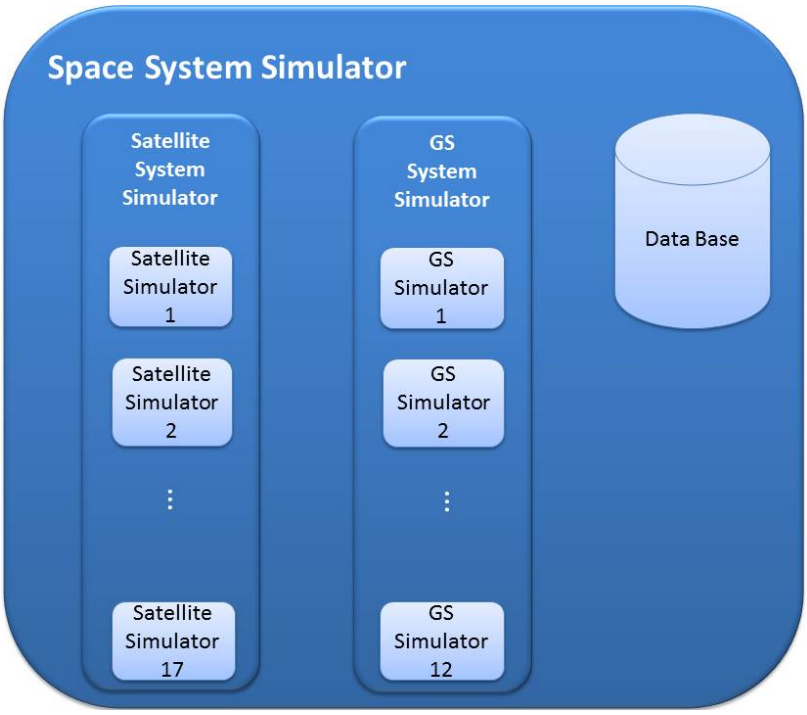


Figure 5.10: Space System Simulator's Architecture

- Finally, when the scenario finishes, the simulation is stopped.

The *Database*, the *Satellite System Simulator* and the *Ground Station System Simulator* are thoroughly described in the next subsections.

5.2.4.1 Database

A database containing all the required data for the simulations was implemented. The selected database management system *(DBMS)* is *MySQL*. It was selected because it is friendly usable, works in multiple platforms, provides transactional and nontransactional storage engines and the server is a separate program for use in a *client/server* networked environment.

The design of the database architecture is shown in Figure 5.11.

The database is constituted by three tables: *Scenarios table*, *Satellites table* and *GroundStations table*. The functionalities of the tables are described in Table 5.7 :

5.2.4.2 Database Filling

- **Filling during initialization**

  The database is filled during the initialization process by executing *setDatabase.py* as previously described in Section 5.2.3.3. This execution makes the following sequential actions:

  1. All the data that is in the database is cleaned to avoid inconsistencies.

Table	Function	Columns	Relationship
Scenarios table	It contains the name of the scenario, the start time T_0 and the end time T_f. Its primary key[1] (represented with a key symbol in Figure 5.11) is the name column.	name timeIni timeEnd	None
GroundStations table	It contains the ID of the Ground Station and its name. Its *primary key* is the idGroundStation.	idSatellite scenario idGroundStation timeInStation timeOutStation interestZoneIni interestZoneEnd	None
Satellites table	It contains the ID of the satellites *idSatellite*, the scenario and the ground station id, *idGroundStation*, in which the events occur, the time in which the satellite enters into the visibility cone $[T_{GS}]_{0ij}$, *timeInStation*, the time when it leaves the visibility cone $[T_{GS}]_{fij}$, *timeOutStation*, the time when the satellite enters into the *AIO* $[T_{AOI}]_{0i}$, *interestZoneIni*, and the time when the satellite leaves the *AOI* $[T_{AOI}]_{fi}$,*interestZoneEnd*. All of them are primary keys because a satellite can download images of several *AOIs* in a single ground station.	idGroundStation name ip port	This table relates the *Scenarios* table and the *GroundStations* table. This is because the *scenario* and *idGroundStation* columns are foreign keys[2]. The characteristics of these columns are *On delete cascade* and *On update cascade*. This means that when an object contained in a table (*GroundStation* table or *Scenarios* table) is deleted, it is also automatically deleted the object that refers to it, and if it is updated the object is also automatically updated.

Table 5.7: Funcionalities of the database tables for the simulator

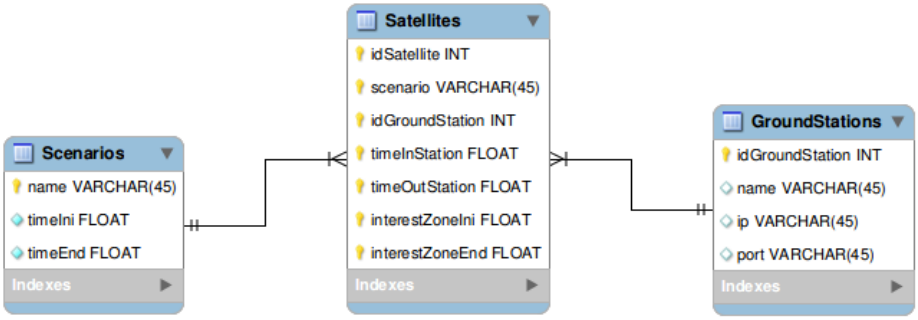


Figure 5.11: Database architecture

2. The *GroundStation* table is filled with the ground stations information (only name and id).
3. The Scenario table is completely filled with the data of each scenario.
4. Finally, the Satellites table is completely filled. This process joins the information contained in the *Scenario_<NUM>_<SCENE>.csv* with file *All_Scenarios.csv*. This means that for each Scenario the $[T_{GS}]_{0ij}$, $[T_{GS}]_{fij}$, $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$ representative of the simulated scenario are taken. That information is inserted into the database as follows:
   - For each scenario the information regarding the accesses of the satellites to the ground stations are selected, $[T_{GS}]_{0ij}$ and $[T_{GS}]_{fij}$.
   - Those accesses are merged with the *All_scenarios.csv* file in order to obtain the accesses of the satellites in which there are *AOI* images.
   - Then if the satellite acquired an *AOI* in the current scenario, the information of the *AOI* is also included in the database.
   - Otherwise the fields with the information regarding the *AOI* are filled with -1 value.

- **Filling during the execution of the Space System Simulator**

  During the execution of the *Ground Station System Simulator* the ip and port columns in the *GroundStations* table are filled. This is done when a *Ground Station Simulator* starts its execution in a node. The *IP adress* and *port* of that node are obtained and included in the database. Once each ground station has an *IP adress* and a *port* associated, the *Satellite Simulators* can read both of them and communicate with *Ground Stations Simulators* servers.

5.2.5 Satellite System Simulator

The *Satellite System Simulator* reproduces the behaviour of the 17 satellites constellation by executing 17 individual *Satellite Simulators* characterized by the specific behaviour of each satellite.

The *Satellite System Simulator* is a manager that executes 17 *Satellite Simulators*. The *Satellite System Simulator* requires the scenario number and the *IP adress* of the database as an input and executes the *Satellite Simulators* by providing them the specific identity of the satellite *(idSatellite)*, number of the scenario *(S)* and IP of the distributed database *(ipDatabase)*.

Next, the architecture of the *Satellite Simulator* for each individual satellite is described.

5.2.5.1 Satellite Simulator

The *Satellite Simulator* is constituted by the following components:

- *Initialization Module:* This module obtains the following parameters and initializes the Satellite Simulator:
  - *S:* the number of the scenario to simulate.
  - *idSatellite:* identification number of the satellite (1 to 17).
  - *ipDatabase: IP address* of the database.

  Once obtained the previous parameters, the Initialization Module connects with the database and fetches the $[T_{GS}]_{0ij}$, $[T_{GS}]_{fij}$, $[T_{AOI}]_{0i}$, $[T_{AOI}]_{fi}$, the *IP addresses* of the *Ground Station Simulators* servers "ipGroundStations" and the ports of the *Ground Station Simulators* servers "portGroundStations" for the simulated satellite in the specific scenario. Those times$[T_{GS}]_{0ij}$, $[T_{GS}]_{fij}$, $[T_{AOI}]_{0i}$, $[T_{AOI}]_{fi}$ are provided to the *Satellite Dynamics Module*.

- *Satellite Dynamics Module:* This module represents the dynamics and associated parameters of the satellite. It requires $[T_{GS}]_{0ij}$, $[T_{GS}]_{fij}$, $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$ of every scenario as inputs, and it provides the acquired images by the satellite in function of the time acquisition (those images are downloaded to the ground stations, simulated by the *Ground Station System Simulator*). Figure 5.12 shows the relation between the previous modules.

5.2.5.1.1 Satellite Simulator Workflow

The *Satellite Simulator* starts the execution following these steps:

1. The *Initialization Module* obtains *S*, *idSatellite* and *ipDatabase*.
2. Then, the *Initialization Module* makes queries on the database to obtain all the required satellite data for current scenario: $[T_{GS}]_{0ij}$, $[T_{GS}]_{fij}$, $[T_{AOI}]_{0i}$ and $[T_{AOI}]_{fi}$. The queries made to the database contain the particle "order by timeInZone" column. This means that the database returns the $[T_{GS}]_{0ij}$ times in ascending order. Other query gets the *ipGroundStations* and *portGroundStations* of every *Ground Station Simulator*.

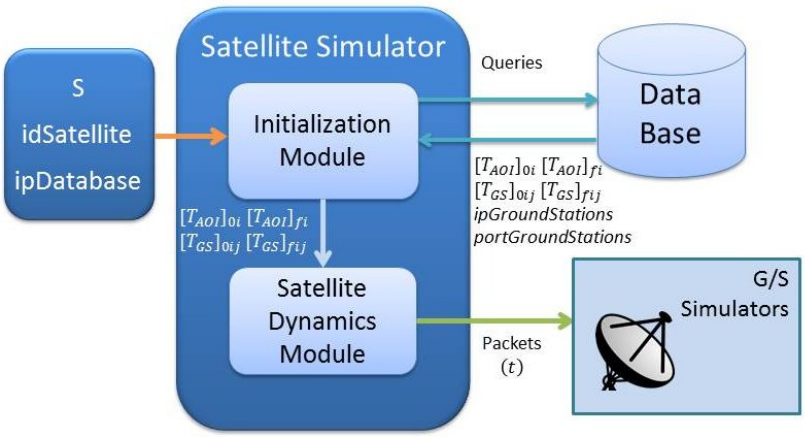


Figure 5.12: Satellite Simulator Architecture

3. Schedule of data download: the *Satellite Simulator* schedules the data download from the *Satellite Simulator* to the *Ground Station Simulators* (see Section 5.2.5.1.1.1).

4. When all the communications between the *Satellite Simulator* and the *Ground Station Simulators* have been scheduled in time, the *Satellite Simulator* starts *Satellite Dynamics Module*.

5. When the last scheduled task has been executed, the *Satellite Simulator* finishes the execution.

Figure 5.13 shows the process explained above. The diagram also shows the interactions with the other subsystems (see also Figure 5.12). Figure 5.13 also shows the inputs for the *Satellite Simulator* and the outputs after its execution. These outputs are the packets in time sent to the ground stations and a log file that contains the information about the execution. In orange the inputs to the *Satellite Simulator* are represented, in green the transitions in the workflow and in blue the outputs.

5.2.5.1.1.1 Schedule of data download process

Definitions:

Regarding the images download process, the satellites should download at $160Kbps$ to the ground station. In *Virtual Wall* we have a limited bandwidth at $100Kbps$. To solve this problem we designed several types of packets (with one byte size), which are sent between *Virtual Wall* nodes instead of real satellite imagery. The packets contain all the metadata of the images: *AOI*, non *AOI*, area between visibility cones. These packets are the following:

- *Packet "U":* The transmission of this packet means that the satellite sends *AOI* images

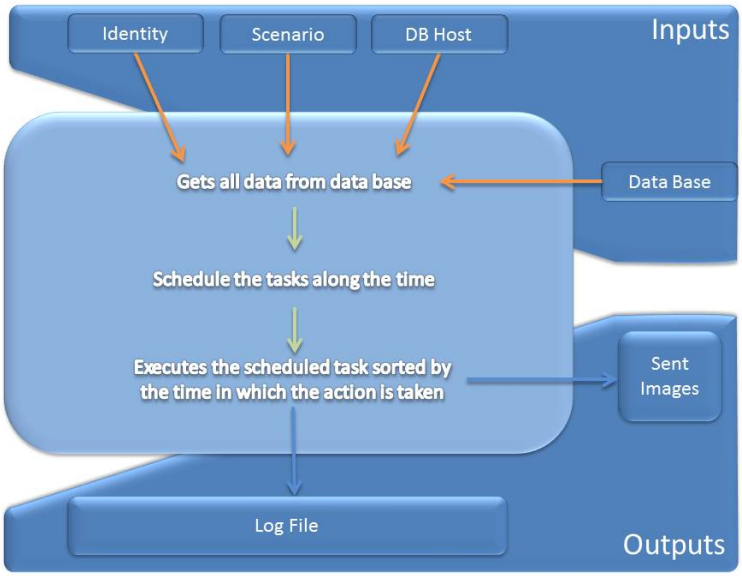


Figure 5.13: Satellite Simulator Workflow

at $98.6Mbps$ and *non AOI* images at $64.1Mbps$.

- *Packet "I":* The transmission of this packet means that the satellite sends *non AOI* images at $160Mbps$.
- *Packet "B":* The transmission of this packet means that the satellite sends *AOI* images at $160Mbps$.

To schedule the data download process the following functions were defined:

- *NotInterestingZone:* it represents the non *AOI* area that is being recorded by the satellite when it is inside a visibility cone. It creates a new connection with the *Ground Station Simulator* and transmits packets type "I". The inputs of this function are the following:
  - *ipGroundStation: IP address* of the *Ground Station Simulator*.
  - *t_ini:* start time in which the function is executed.
  - *t_end:* end time in which the function is finished.

  The function has implemented the pseudo-code in Listing 5.3:

- *InterestingZone:* It represents the *AOI* area that is being recorded by a satellite when it is inside a visibility cone. It creates a new connection with the *Ground Station Simulator* and transmits packets type "U" and "B". The inputs of this function are the following:
  - *ipGroundStation: IP address* of the *Ground Station Simulator*.
  - *t_ini:* start time in which the function is executed.

```
function NotInterestingZone (t_ini,t_end,ipGroundStation):
offset <-0 //deviation of the normal time
socket <- connect(ipGroundStation) //connection with the GS
current_t <- time() //get the current system time
penal_times <-0 // times that time function is called
time_penalty <- time that time() costs
while (current_t  <  t_end + offset):
        Socket.send('I')
        Penal_times <- penal_times+2
        Offset = penal_times * time_penalty
        Time.sleep(0.2-(time()-t_temp))
        t_temp <- time()
end while
socket.close()
```

Listing 5.3: Pseudocode of *NotInterestingZone* function

- *t_end:* end time in which the function is finished.
- *offset_time:* It is the difference $([T_{GS}]_{0ij} - [T_{AOI}]_{0i}) * CR$.

The function has implemented the pseudo-code in Listing 5.4:

```
function InterestingZone (t_ini,t_end,offset_time,ipGroundStation):
offset <-0 //deviation of the normal time
socket <- connect(ipGroundStation) //connection with the GS
current_t <- time() //get the current system time
penal_times <-0 // times that time function is called
time_penalty <- time that time() costs

while (current_t <  t_ini+offset_time + offset):
        socket.send('B')
        penal_times <- penal_times+2
        offset = penal_times * time_penalty
        time.sleep(0.2-(time()-current_t))
current_t<- time()
end while
while (current_t <  t_end + offset):
        socket.send('U')
        penal_times <- penal_times+2
        offset = penal_times * time_penalty
        time.sleep(0.2-(time()-current_t))
current_t<- time()
end while
socket.close()
```

Listing 5.4: Pseudocode of *InterestingZone* function

- *OutOfVisibility:* The satellite is not in the visibility cone of a ground station, the satellite follows the orbit until it enters into a visibility cone. It does not transmit any packet. The inputs of this function are the following:
  - *t_ini:* start time in which the function is executed.
  - *t_end:* end time in which the function is finished.

The function has implemented the pseudo-code in Listing 5.5:

```
function OutOfVisibility (t_ini,t_end):
offset <−0 //deviation of the normal time
socket <− connect(ipGroundStation) //connection with the GS
current_t <− time() //get the current system time
penal_times <−0 // times that time function is called
time_penalty <− time that time() costs
while (current_t  <  t_end + offset):
        penal_times <− penal_times+2
        offset = penal_times * time_penalty
        time.sleep(0.2−(time()−t_temp))
        t_temp <− time()
end while
socket.close()
```

Listing 5.5: Pseudocode of *OutOfVisibility* function

Scheduling process:

This process consists of the scheduling all the satellite interactions with the ground stations. This is done as follows:

1. If the area the satellite is flying over is not *AOI*, the function *NotInterestingZone* is scheduled to be executed as follows: *NotInterestingZone(ipGroundStation,* $[T_{GS}]_{0ij}$*,* $[T_{GS}]_{fij}$*)*. The function *OutOfVisibility* is scheduled to be executed as follows: *OutOfVisibility(*$[T_{GS}$ $[T_{GS}]_{(0i(j+1)})$*)*, and the scheduling process finishes.
2. Otherwise three cases can take place:
   - If $[T_{AOI}]_{0i} < [T_{GS}]_{0ij} || [T_{AOI}]_{0i} = [T_{GS}]_{0ij}$ the function *InterestingZone* is scheduled to be executed as follows: *InterestingZone(ipGroundStation,* $[T_{GS}]_{0ij}$*,* $[T_{AOI}]_{fij}$ *, offset_time)*. It is always considered that $[T_{AOI}]_{fi} > [T_{GS}]_{fij}$.
   - If $[T_{AOI}]_{0i} > [T_{GS}]_{0ij}$, first, the function *NotInterestingZone* is scheduled to be executed as follows: *NotInterestingZone(ipGroundStation,* $[T_{GS}]_{0ij}$*,* $[T_{AOI}]_{0ij}$*)*; and second, the function *InterestingZone* is scheduled to be executed as follows: *InterestingZone(ipGroundStation,* $[T_{AOI}]_{0ij}$*,* $[T_{AOI}]_{fij}$*, 0)*.
3. The *NotInterestingZone* function is scheduled to be executed as *NotInterestingZone(ipGroundStatio* $[T_{AOI}]_{fij}$*,* $[T_{GS}]_{fij}$*)*.
4. The function *OutOfVisibility* is scheduled to be executed as follows: *OutOfVisibility(*$[T_{GS}]_{fij}$*,* $[T_{GS}]_{(0i(j+1)})$*)*, and the scheduling process finishes.

Figure 5.14 graphically shows the scheduling process explained above.

The activity diagram in *UML* format of the *Satellite Simulator* is depicted in Figure 5.15:

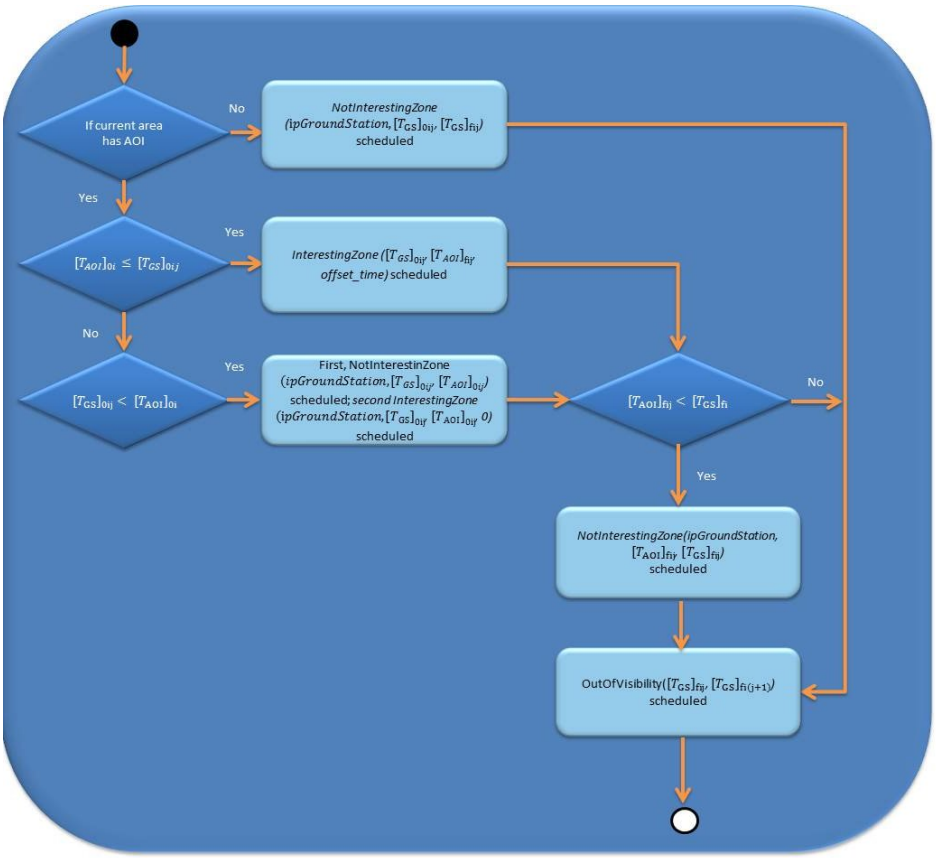


Figure 5.14: Sheduling Process on the Satellite Simulator

5.2.5.1.2 Satellite Simulator Implementation

The implementation was done in Python2.7. The python's libraries needed to implement the software are listed in Table 8.

When we installed Python, all the libraries were installed with the exception of "MySQLdb", which was manually installed.

Furthermore, STK provided the time in the next formats:

- *UTCG format:* Universal Time Coordinated in Gregorian format.
- *Format in seconds:* total of seconds

They are the absolute time from the beginning of the constellation simulation in the STK software. In the data base, the stored times are implemented in seconds. However to simulate a scenario in real time is too long. It can be shortened by computing a relative time (T_r):

$$T_r = T/n \tag{5.10}$$

where T is the absolute time and n a factor that scales the absolute time to reduce the

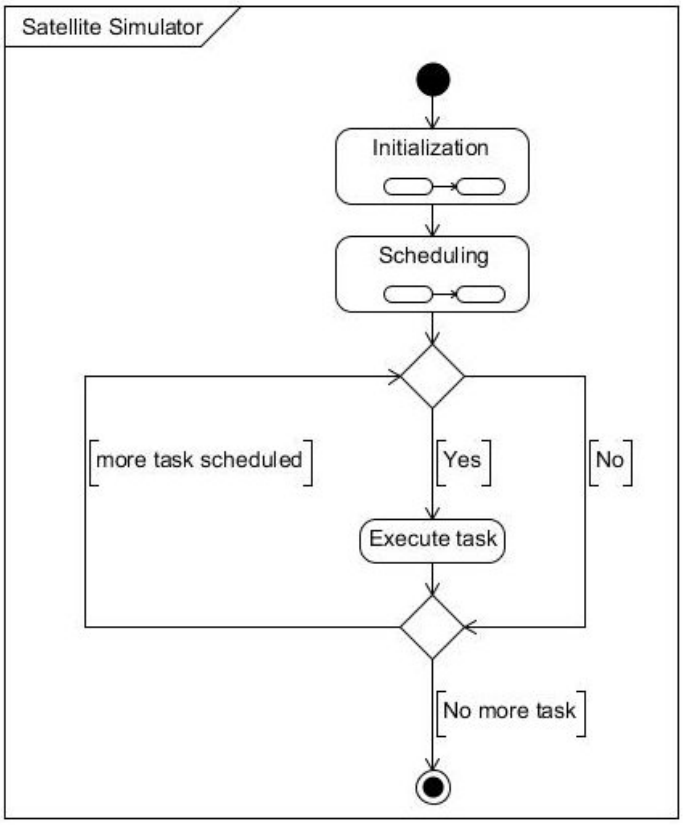


Figure 5.15: Satellite Simulator Activity Diagram

execution time of the experiment. In preliminary simulations we used $n = 10$, although during the experiment execution stage this will be evaluated and probably changed. The Table «»shows the absolute times and relative times for each scenario duration.

5.2.5.1.3 Execution

To execute the *Satellite Simulator Software* the following dependencies are required:

- Operative System based in *GNU Debian*.
- Python v.7
- Python packages: (shows in Table 8).
- Ethernet interface for the network connection.
- Connectivity with the data base located in *BonFIRE* through the network.

The *Satellite System Simulator* software is developed in a multiplatform language, but it is restricted to be executed in *UNIX* operative systems because there are many dependencies with some packages and the file system.

The execution of the satellite software has to be done as *super-user* in *UNIX*. It is executed

Python Library	Function
Sys	System library
OS	Operative system interactions supply
Sched	Library that allow the Satellite Simulator to schedule the task along the time
Time	For managing the time
Socket	Library for creating and establishing connections with other host
Pdb	Used for debugging the software
Logging	Log

Table 5.8: Satellite Simulator's Python Libraries

Scenario	Absolute Time (T_0, T_f) (Seconds)	Relative Time T_{r0}, T_{rf} (Seconds)
Scenario 1: Emergencies – Lorca Earthquake (Spain)	(63638, 63661)	(6363.8, 6366.1)
Scenario 2: Infrastructure monitoring. Affection in railway infrastructures by sand movement in desert areas (Spain)	(54060, 54753)	(5406.0, 5475.3)
Scenario 3: Land Management – South West of England	(62480, 63865)	(6248.0, 6386.5)
Scenario 4: Precision Agriculture – Argentina	(78461, 84999)	(7846.1, 8499.9)
Scenario 5: Basemaps – Worldwide	(0, 86400)	(0, 8640.0)

Table 5.9: Scenarios relative times

with the following command line:

> python satellite.py <IDSAT> <SCENARIO> <DBHOST> [LOGLEVEL]

where:

- *IDSAT* is the satellite identity. This value must be an integer.
- *SCENARIO* is the scenario number to simulate. Must be an integer.
- *DBHOST* is the host where the data base is located. Must be a hostname or an *IP address*.
- *LOGLEVEL* is the level of log that the software will show. The values can be: INFO, DEBUG. This parameter is optative. By default its value is INFO.

5.2.6 Ground Station System Simulator

In this section, the implementation and the architecture of the *Ground Station Simulator* is described. The *Ground Station System Simulator* reproduces the behaviour of the set of 12 ground stations by replicating 12 times a *Ground Station Simulator* that renders individual ground stations behaviour.

The *Ground Station System Simulator* is a manager that executes 12 *Ground Station Simulators*. The *Ground Station System Simulator* requires the scenario number and the *IP address* of the database as an input and executes the *Ground Station Simulators* by providing them the specific identity of the ground station *(idGroundStation)*, number of the scenario *(S)* and *IP address* of the distributed database *(ipDatabase)*.

5.2.6.1 Ground Station Simulator

The functions of the ground stations are the following:

- To receive the images sent by the satellites.
- To create the files with the raw images that the *Orchestrator* will download for processing and publishing in the *Archive and Catalogue* subsystem.

The *Ground Station Simulator* software is common for all the ground stations, but it is parameterized to define each of them, similarly to the *Satellite Simulator*.

The *Ground Station Simulator* is constituted by the following components:

1. *Initialization Module:* This module obtains the following parameters and initializes the *Ground Station Simulator*:
   - *S:* the number of the scenario to simulate.
   - *idGroundStation:* identification number of the ground station (1 to 12).
   - *ipDatabase: IP address* of the database.

Once obtained the previous parameters, the *Initialization Module* connects with the database and fetches the name of the ground station. Also, a query to update the *IP address* and port of the *Ground Station Simulator* server is sent to the database.

2. *Ground Station Dynamics Module:* This module represents the dynamics and associated parameters of the ground station. It requires *S*, *ipDatabase* and *idGroundStation*.

Figure 5.16 shows the relation between the previous modules.

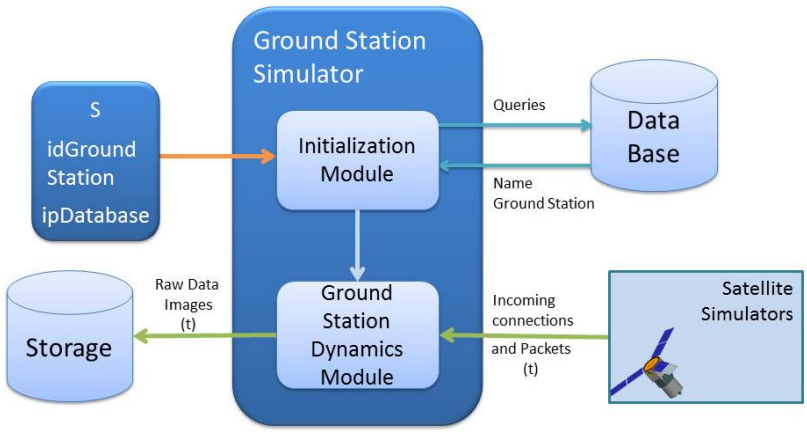


Figure 5.16: Ground Station Simulator Architecture

5.2.6.2 Ground Station Simulator Workflow

The *Ground Station Simulator* starts the execution following these steps:

1. The *Initialization Module* obtains *S*, *idGroundStation* and *ipDatabase*.
2. Then, the *Initialization Module* queries on the database for the name of the ground station and for updating the *IP address* and *port* of the *Ground Station Simulator*.
3. The *Ground Station Dynamics Module* starts:
   - A network socket[3] is created for listening input connections from *Satellite Simulators*.
   - Every time a satellite enters into the visibility cone of the ground station a connection between a *Satellite Simulator* and the *Ground Station Simulator* is created. Note that if several satellites are in the same visibility cone, the *Ground Station Simulator* opens a connection for each *Satellite Simulator*, and that the *Ground Station Simulator* can listen to new connections if new satellites enter into the visibility cone.

[3]Network socket is an endpoint of an inter-process communication flow across a computer network. Most of the communications between computers is based on the Internet Protocol using network sockets.

- The *Ground Station Simulator* creates a new process to keep the previous connection or connections (if there are more than one satellite in the visibility cone) open.
- The new process or processes receive the packets sent by the *Satellite Simulators*.
- Once the transmission between the *Satellite Simulators* and the *Ground Station Simulator* finished the connections are closed.
- Then the received packets are counted and classified by type (see Section 5.2.5.1.1.1):
  - *Packet "U":* $98.6Mb$ are added to the *AOI* buffer and $64.1Mb$ are added to the *non AOI* buffer in the *Ground Station Simulator*.
  - *Packet "I":* $160Mb$ are added to the *non AOI* buffer in the *Ground Station Simulator*.
  - *Packet "B":* $160Mb$ are added to the *AOI* buffer in the *Ground Station Simulator*.

  After classifying the packets, the new processes finish.
- The *AOI* images and *non AOI* images are created in the *Ground Station Simulator*.

4. The *Ground Station Simulator* server ends when it receives the *SIGINT* signal.

Figure 5.17 shows the process explained above. The diagram also shows the interactions with the other subsystems (see also Figure 5.16). Figure 5.17 also shows the inputs for the *Ground Station Simulator* and the outputs after its execution. These outputs are the Raw Images in time created and a log file that contains the information about the execution.

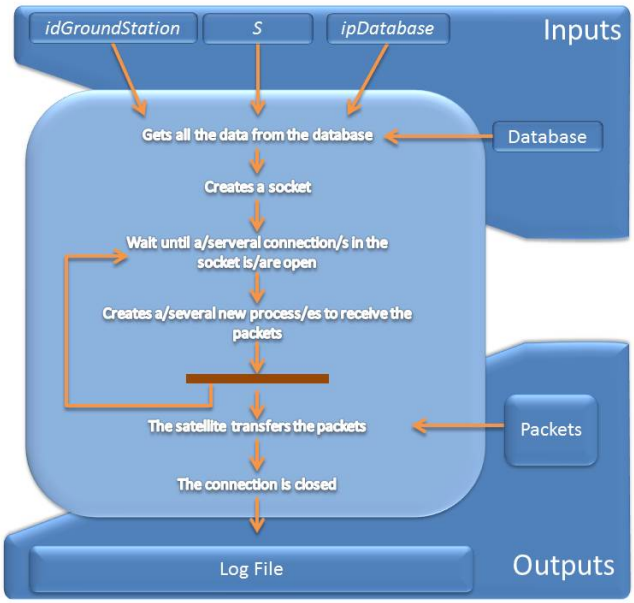


Figure 5.17: Ground Station Simulator Workflow

The activity diagram in UML format of the Ground Station Simulator is depicted in Figure 5.18.

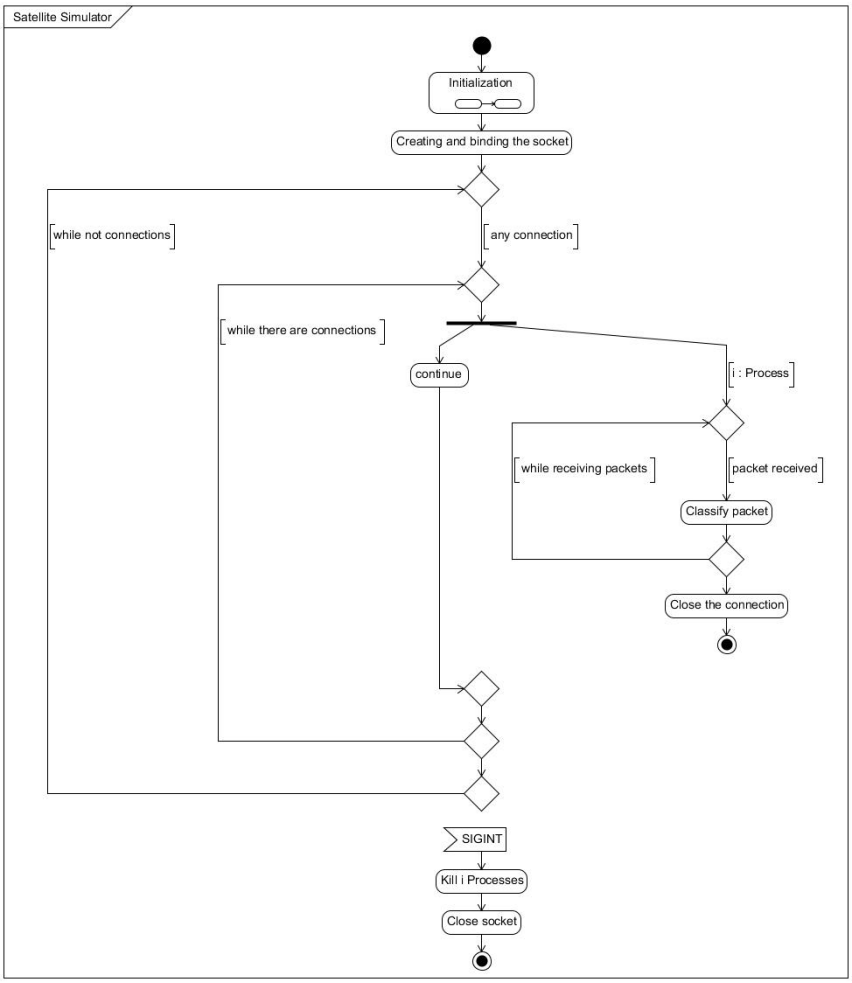


Figure 5.18: Ground Station Simulator Activity Diagram

5.2.6.2.1 Implementation

The implementation was done in Python2.7, using the libraries described in Table 5.10.

5.2.6.2.2 Execution

To execute the Ground Station Simulator Software the following dependencies are required:

- Operative System based in *GNU Debian*.
- Python v.7.

- Python packages: (shows in Table 5.10).
- Ethernet interface for the network connection.
- Connectivity with the data base located in *BonFIRE* through the network.

The *Ground Station Simulator* is developed in a multiplatform language, but it is restricted to be executed in *UNIX* operative systems because there are many dependencies with some packets and the file system.

The software also needs to know the *IP addresses* assigned to its Ethernet interface. If this interface is not connected, the software looks for the WLAN0 interface. This information is accomplished from the *UNIX* file "/etc/hosts".

The execution of the satellite software must be done as sudo user as follows:

> python groundstation.py <IDSAT> <SCENARIO> <DBHOST> [LOGLEVEL]

where:

- *IDSAT* is the ground station identity. This value must be an integer.
- *SCENARIO* is the scenario to simulate. Must be an integer.
- *DBHOST* is the host where the data base is located. Must be a hostname or an *IP address*.
- *LOGLEVEL* is the level of log that the software will show. The values can be: INFO, DEBUG. This parameter is optative. By default its value is INFO.

5.3 Implementation in Virtual Wall

In this section, the implementation is introduced indicating the testbeds involved in it and the required steps for the implementation.

Fistly, the implementation of the *Space System Simulator* in *Virtual Wall* is depicted. The designed topology and the new modified topology (different to the designed one) are depicted because of the handicaps of the hardware during the implementation. Then, nodes reservation is explained and detailed. Finally, the scripts included in each type of node are described before the presentation of the execution.

The next section is dedicated to the implementation of the software developed in the *BonFIRE* testbed. Nodes reservation is firstly depicted. Architecture and setup are also explained. And again, the scripts for each node setup are included before the description of the execution.

Finally, the integration between *Virtual Wall* and *BonFIRE* is described.

5.3.1 Implementation in Virtual Wall

In *Virtual Wall* the *Space System Simulator* was implemented. It is constituted of the following modules:

- The *Satellite System Simulator*
- The *Ground Station System Simulator*

The topology network designed is depicted in Figure 5.19.

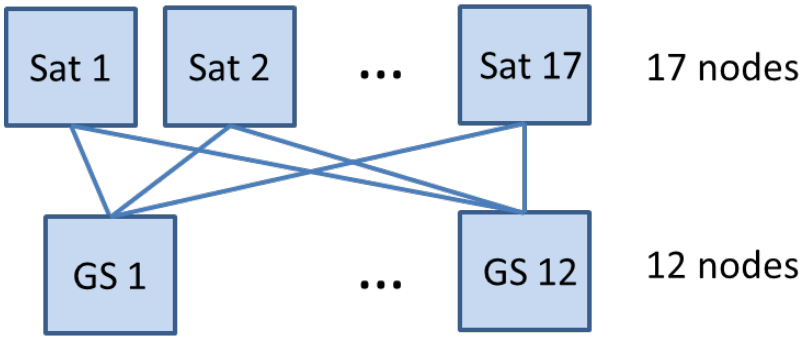


Figure 5.19: Topology Network in *Virtual Wall*

The previous topology network involves Sat nodes to have 12 connections with the GS nodes; and GS nodes 17 connections with Sat nodes plus one connection with the BonFIRE cloud.

The nodes in *Virtual Wall* have a limitation of 5 physical connections. To deploy the previous topology we adapted the *Space System Simulator* software to establish those connections by software. This was done by making the *Satellite System Simulator* flexible to connect with any ground station from GS1 to GS12. Then the *Satellite System Simulator* can be multiplied and only connect to it the number of GS nodes desired.

The solution was to multiply the *Satellite System Simulator* by three and connect 4 ground stations to each *Satellite System Simulator* . This ensures the same performance and dynamics of the previous topology network described in Figure 5.19, and the only thing it changes is the software.

The *Space System Simulator* implemented in *Virtual Wall* is depicted in Figure 5.20.

5.3.1.1 Nodes Reservation and Setup

The *Space System Simulator* is then constituted by 15 “Generic nodes” in *Virtual Wall* 1 (The *Virtual Wall* testbed is composed by two sets of machines, *Virtual Wall* 1 and *Virtual Wall* 2). The configuration of the experiment makes the provisioning of “any available node”, which is more flexible in case a node is being used in other experiment (see Figure 5.21). The links between SS *(Satellite Simulators)* and GS *(Ground Stations Simulators)*

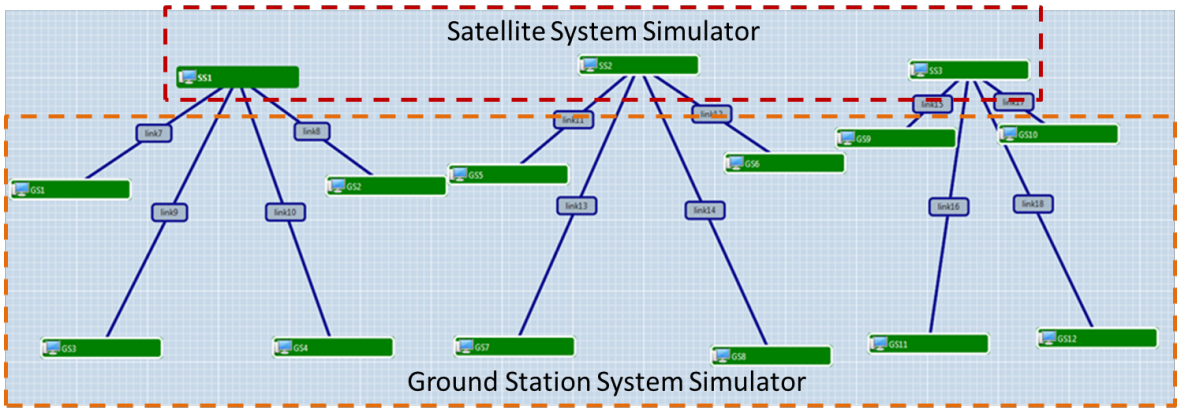


Figure 5.20: Topology Network in *Virtual Wall*

nodes are *TCP*. The definition of the dependencies and the commands that will be installed at setup is defined in *JFed* using a *RSPEC* specification. This specification includes some bash commands or instructions that will execute after reservation automatically.

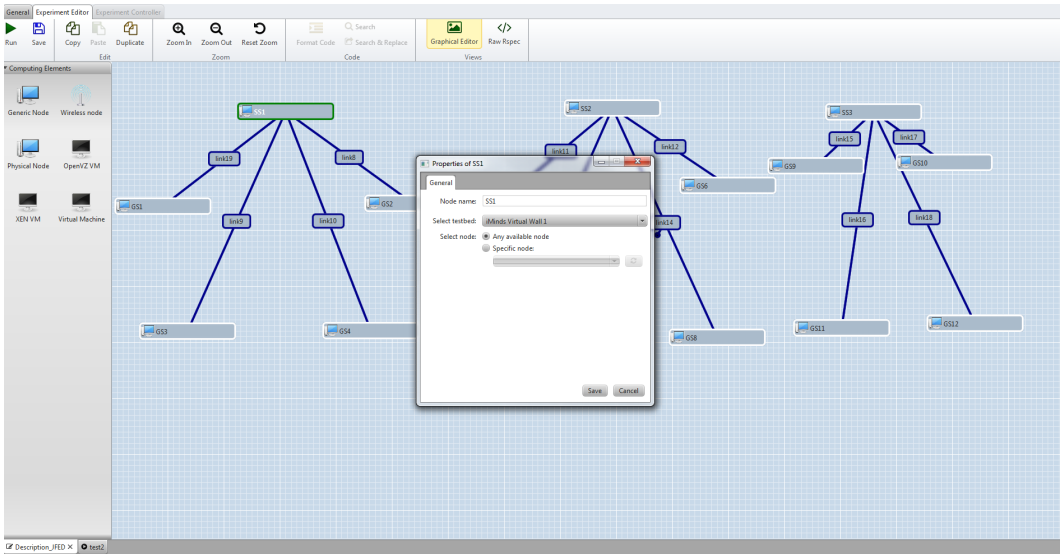


Figure 5.21: Configuration of *Virtual Wall* nodes

5.3.1.1.1 Satellite Simulator Nodes Setup

The configuration of the nodes is done by using *JFed RSPEC Experiment*. The setup steps are the following:

- To configurate a gateway for obtaining Internet connectivity and installing the needed libraries.
- To update the system.

- Once the update is finished, dependencies can be installed.
- As the simulators require a connection with a database, the *MySQL* library is also required.
- Source is downloaded from Google Drive where is located.

The script that perform the above is showed in Listing *code:impl-rspec-sat-sim*.

```
<execute shell="sh" command="sudo route del default gw 10.2.15.254"/>
<execute shell="sh" command="sudo route add default gw 10.2.15.253"/>
<execute shell="sh" command="sudo apt-get update"/>
<execute shell="sh" command="sudo apt-get install python python-
   mysqldb -y"/>

<execute shell="sh" command="sudo wget --no-check-certificate <<
   google_drive_host_address>> -O
/users/jbecedas/push_dbIP.sh"/>
<execute shell="sh" command="sudo bash /users/jbecedas/push_dbIP.sh
   129.215.175.147"/>
<execute shell="sh" command="sudo wget --no-check-certificate
<<google_drive_host_address>> -O /users/jbecedas/satellite.py"/>
```

Listing 5.6: RSPEC specification for *Satellite Simulators*

Where «google_drive_host_address» is the address of the Google Drive URI in which the scripts are located.

The *Database* is located in *BonFIRE* and it has an *IP address* that can change. In order to obtain that *IP address* and include it in the *Satellite Simulator*, a simple script to acquire the *IP address* of the database was developed *push_dbIP.sh*. Listing 5.7 contains the script.

```
#!/bin/bash

ip=$1

touch /users/jbecedas/ipdb
echo $ip > /users/jbecedas/ipdb
```

Listing 5.7: Bash script to write the Database's*IP address* on a file

The execution of the *push_dbIP.sh* script acquires the *IP address* of the database and includes it in the file *ipdb*, created by *push_dbIP.sh*:

Now, the node is perfectly configured to proceed with the installation of the *Satellite Simulator*. The software of the simulator has been archived too in *Google Drive*. With the following command the software is acquired:

It can now be executed in the node.

5.3.1.2 Ground Station Simulator Nodes Setup

The *Ground Station Simulators* software is developed under Python. The steps of setup are the following:

- To configurate a gateway for obtaining Internet connectivity and installing the needed libraries.
- To update the system.
- Once the update is finished, dependencies can be installed.As occurred with the SS nodes, the update and the installation of Python and MySQL are required.
- As the simulators require a connection with a database, the *MySQL* library is also required.
- Source is downloaded from Google Drive where is located.
- The database IP address is added to *ipdb* file.
- The raw data is located in Google Drive too.
- The GS nodes are required to open an *FTP* to be accessed from the *Orchestrator* in *BonFIRE* . The *FTP* is configured by executing the *install_ftp.sh* file. This file content is in Listing 5.9.

The script that perform the above is showed in Listing *code:impl-rspec-gs-sim*.

```
<execute shell="sh" command="sudo route del default gw 10.2.15.254"/>
   <execute shell="sh" command="sudo route add default gw
   10.2.15.253"/>
<execute shell="sh" command="sudo apt-get update"/>  <execute shell="
   sh" command="sudo apt-get install python python-mysqldb -y"/>
<execute shell="sh" command="sudo wget --no-check-certificate
   google_drive_host_address -O /users/jbecedas/push_dbIP.sh"/> <
   execute shell="sh" command="sudo bash /users/jbecedas/push_dbIP.sh
    129.215.175.147"/>
<execute shell="sh" command="sudo wget --no-check-certificate
   google_drive_host_address
-O /users/jbecedas/groundstation.py"/>
<execute shell="sh" command="sudo wget --no-check-certificate
   google_drive_host_address -O
/tmp/original.bin "/>
```

Listing 5.8: RSPEC specification for *Ground Stations Simulators*

Through the FTP connection the raw data will be transferred from the GS nodes to the BonFIRE cloud.

The execution of the Satellite Simulators and the Ground Station Simulators are described in Section 5.2.5.1.3 and Section 5.2.6.2.2.

```
#!/bin/bash

sudo apt-get update
sudo apt-get install mysql-client ftp ftplib3 vsftpd -y
sudo sed -i "s/listen=YES/listen=NO/" /etc/vsftpd.conf
sudo sed -i "s/#listen_ipv6=NO/listen_ipv6=YES/" /etc/vsftpd.conf
sudo sed -i "s/#listen_ipv6=YES/listen_ipv6=YES/" /etc/vsftpd.conf
sudo sed -i "s/anonymous_enable=YES/anonymous_enable=NO/" /etc/vsftpd.
   conf
sudo sed -i "s/#local_enable=NO/local_enable=YES/" /etc/vsftpd.conf
sudo sed -i "s/#local_enable=YES/local_enable=YES/" /etc/vsftpd.conf
sudo sed -i "s/#write_enable=NO/write_enable=YES/" /etc/vsftpd.conf
sudo sed -i "s/#write_enable=YES/write_enable=YES/" /etc/vsftpd.conf
sudo sed -i "s/#local_umask/local_umask/" /etc/vsftpd.conf
sudo sed -i "s/#chroot_list_file/chroot_list_file/" /etc/vsftpd.conf
sudo mkdir -p /home/ftp/deimos
sudo mkdir /home/deimos
sudo useradd -d /home/deimos -g ftp  deimos
sudo chmod 777 -R /home
sudo touch /etc/vsftpd.chroot_list
sudo su
echo "deimos:deimos"|chpasswd
echo "deimos" >> /etc/vsftpd.chroot_list
sudo service vsftpd restart
```

Listing 5.9: FTP server installation

5.4 Cloud Architecture

In this section, the EO architecture on cloud is fully described. As mentioned, the cloud infrastructure where this system is implemented is provided by *BonFIRE* multi-cloud. The *Orchestrator*, *Archive and Catalog* and *Processing Chain* integrate the cloud components for processing, archiving and cataloging Earth's surface images. This modules are summarized as follows:

- *Orchestrator:* it manages the automatic distribution of the raw data to the processors. It handles the complete automatic processing chain execution. If the processor chain is occupied, the manager replicates the complete chain in a new machine.
- *Processing Chain:* They process the raw data and convert it in orthorectified images.
- *Archive and catalog:* It is the place where the processed images are stored and cataloged for its distribution.

In the following sections, these cloud components are depicted and the interactions among them explained also. First, the product processors compose the *Processing Chain* are shown. These processors are owned by Elecnor Deimos company, so they are not fully described but the main features are exposed. Then, the *Archive and Catalog* module is described. Finally, the *Orchestrator* component that manages all the cloud processes is fully explained and detailed.

5.4.1 Processing Chain

The *Processing Chain* is a module that is in charge of processing the payload raw data from the satellites to produce image products. The four, most important operations that the product processors perform on the input data are the following:

- A calibration, to convert the pixel elements from instrument digital counts into radiance units.
- A geometric correction, to eliminate distortions due to misalignments of the sensors in the focal plane geometry.
- A geolocation, to compute the geodetic coordinates of the input pixels.
- An ortho-rectification, to produce ortho-photos with vertical projection, free of distortions.

The previous steps also generate quality-related figures of merit that are made available in all the products. Moreover, the product processors generate metadata, in line with industry standards, to facilitate the cataloguing, filtering and browsing of the product image collection. These components are considerated as black boxes because they are owned by Elecnor Deimos and its designs and implementations can not be published, but them were studied for carrying out this project.

The output image products are classified into four different levels, according to the degree of processing that they have been subjected to (see Figure 5.22):

- *Level 0* products are unprocessed images, in digital count numbers.
- *Level L1A* products are calibrated products, in units of radiance.
- *Level L1B* products are calibrated and geometrically corrected products (ortho-rectified), blindly geolocated.
- *Level L1C* products are calibrated and geometrically corrected products (ortho-rectified), precisely geolocated using ground control points.

5.4.1.1 The L0 Processor

The acquired data is organized into image sectors of predefined size and structure and converted in scenes. Scenes, as defined here, are used throughout the subsequent L1 levels. The size and configuration of the scene is not changed again in the processing chain, for this reason the scene definition is constant for all the L1 levels.

The inputs are the following

- The Raw Data.
- The configuration database.
- The calibration database.

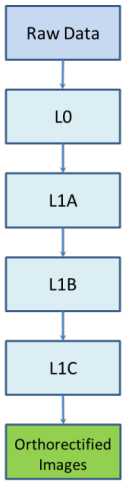


Figure 5.22: Stages of the product processing.

The outputs are the following:

- The L0 products.

5.4.1.2 The L1A Processor

This section describes the functionality of the processors included in the Level 1A of the Automatic Processing Chain. The goal of Level 1A is to calibrate the scenes. The resulting images are given in units of radiances.

The L1A component works on the scenes that compound the L0 product, performing different transformations over pixel values to generate radiances.

The inputs to the L1A level are:

- One L0 scene.
- The configuration database.
- The calibration database.

The output is:

- The L1A product.

5.4.1.3 The L1B Processor

Level 1B implements the geolocation, resampling and packing.

The inputs to the L1B level are the following:

- The L1A product.
- The configuration database.
- The calibration database.

The outputs are the following:

- The L1B products.

5.4.1.4 The L1C Processor

The L1C processor performs the ortho-rectification of the L1B product using ground control points.

The inputs to the L1C level are the following:

- The L1B product.
- The calibration database.
- The configuration database.

The output is the following:

- The L1C products.

5.4.2 Archive and Catalogue

The *Archive and Catalogue* is a shared space of memory between the *Orchestrator*, the product processors and the distribution of data. It has a data acquisition component.

Data Acquisition component: This component manages the input data arriving to the *Archive and Catalog*. The ingestion of data is automatic.

In the *Archive and Catalog* module the processed images are stored and catalogued for their distribution.

The *Archive and Catalog* basically consists of:

- The *Archive* is constituted by optimized storages structure allowing managing a big amount of data, efficient storage and retrieval of any kind of file. The *Archive* shall be organized in hierarchical levels of storage in order to provide a cost effective storage solution.
- The *Catalogue* shall store an inventory database with the metadata of archive files. It allows the product process chain easiness to access to the metadata from the processed products.
- For the added value services the catalog will be accessed by a *Web Service*.
- CSW is a module with the CSW standard for the catalogue (based on OGC standard). For more information on CSW, please refer to OGC *OpenGIS Implementation Specification 07-006r1* and the OGC tutorial on CSW. Through this standard the distribution of data is done.

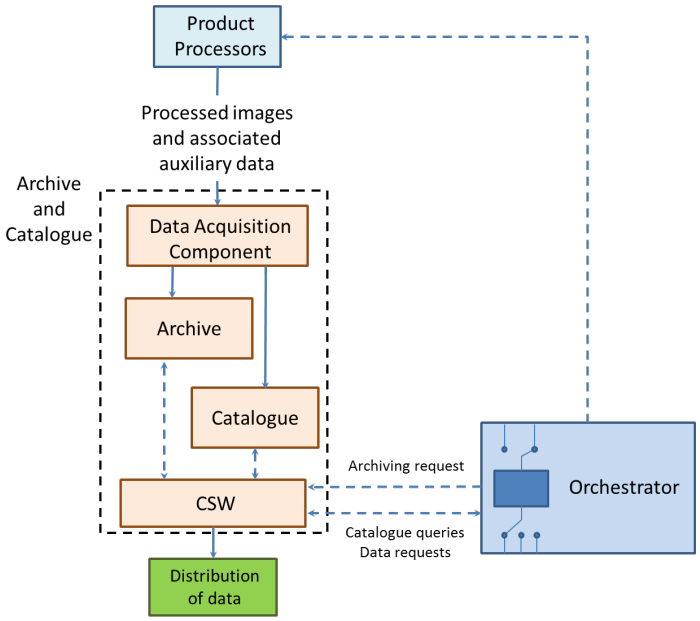


Figure 5.23: Scheme of the Archive and Catalogue module.

5.4.3 Orchestrator

This section describes the design of the *Orchestrator*. The *Orchestrator* is the component that manages the task to be done in the cloud. It is running over the *BonFIRE* Cloud and controls all interactions between all the components implemented in *Fed4FIRE* testbeds as *BonFIRE* and *Virtual Wall* . First, the *Orchestrator*'s funcionalities and the interactions with other components of the GEO-Cloud system are described. Then, the workflow of *Orchestrator* and the interfaces with the *Archive and Catalog* module and the *Processing Chain* module are depicted.

5.4.3.1 Functionality

The *Orchestrator* has the following functions:

- To identify which outputs shall be generated by the processors.
- To generate the Job Orders. They contain all the necessary information that the processors need. Furthermore these XML files include the interfaces and addresses of the folders in which the input information to the processors is located and the folders in which the outputs of the processors have to be sent. They also include the format in which the processors generate their output.
- To look for raw data in the ground stations (pooling) to ingest such raw data in a shared storage unit in the cloud for its distribution to the processing chain.
- To control the processing chain by communicating with the product processors, which have four levels of processing: L0, L1A, L1B and L1C.

- To manage the archive and catalogue.

The orchestrator is designed to be implemented in the GEO-Cloud architecture. It interacts with different modules:

- Ground stations implemented in *Virtual Wall* .
- Processing instances in the cloud.
- Archive and catalogue.

Figure 5.24 depicts the *Orchestrator's* interactions with the other modules of the GEO-Cloud architecture.

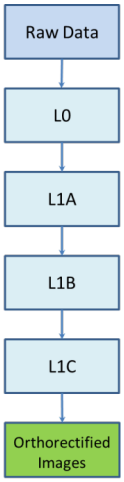


Figure 5.24: Stages of the product processing.

As shown in Figure 5.24, the *Orchestrator* is pooling the Ground Stations frequently. When the *Orchestrator* gets the data, uses the Product Processor for processing the data to generate the result image. When this processing has finished, the *Orchestrator* sends the image to Archive and Catalogue to be available for customers.

5.4.3.2 Workflow

The *Orchestrator* works by following the next sequence of steps:

1. The *Orchestrator* gets all the information about the *Ground Stations Simulators* and localizes them.
2. The *Listener* pools to the *Ground Stations* and when there are a downloadable raw data, the *New_Data_Event* is launched.
3. When the *New_Data_Event* occurs, the *Orchestrator* downloads the data.
4. The *Orchestrator* moves the raw data to a shared storage.

5. Then, the *Orchestrator* makes different *Job Orders* for the processors. The *Job Order* contains all the useful information for the *Product Processors* to proceed with the image processing.
6. The *Orchestrator* gets the *ProcessorChainController* object (this object was made regarding *Singleton pattern*).
7. The *Orchestrator* instructs the *ProcessorChainController* object to create a new processing chain by sending the *JobOrders* created in step 4.
8. The *ProcessorChain Controller* object creates a new *Processing Chain* to process the data.
9. The *Processing Chain* sequentially executes the L0, L1A, L1B, L1C processors.
10. When the *ProcessingChain* has finished, this notifies the *ProcessorChainController* object that the processing ended.
11. The *ProcessingChainController* alerts the *Orchestrator* that the *Processing Chain* has finished.
12. The *Orchestrator* takes the created image and puts it into the *Archive*. Figure 2 depicts the workflow of the *Orchestrator*.

5.5 Implementation in BonFIRE

In this section, the implementation of the software developed in the *BonFIRE* testbed is described. Nodes reservation is firstly depicted. Architecture and setup are also explained. And again, the scripts for each node setup are included before the description of the execution.

5.6 Integration Virtual Wall-BonFIRE

In this section, the junction between the nodes implemented in *Virtual Wall* and the nodes deployed in *BonFIRE* is described.

5.7 Profilling Tool in PlanetLab

In this section, the motivation for the use of *PlanetLab* is explained together with the design of the real system. Then, the platform and tools used are described with their roles in the experiment. In addition, the network and experiment design are broadly discussed. The execution of the experiment is also presented. Finally, conclusions of this implementation are included.

5.7.1 Definitions

- *Effective bandwidth (Mbps)* is the actual bandwidth at which the data can be transmitted on a link. The nominal bandwidth cannot be reached due to network congestion,

the distance between nodes, delays, etc. Effective bandwidth is higher when nodes are closer, the congestion is scarce and the delays in the transmission are not long.

- *Bandwidth of the network(Mbps)*, which is the nominal "width" of the channel used, if the bandwidth increases, more data can simultaneously be sent, reducing the necessary time to transfer a packet of data. It is usually confused with the signal velocity, which affects the time the data takes to travel to the receiver (latency) but bandwidth cannot reduce this time.
- *Loss rate* is the fraction of data lost in the communication with respect to all the data sent. It is a value between 0 and 1. It can also be provided in percentage.
- *Latency (ms)* is the time it takes a signal to travel from its source, trough the communication channel, until it reaches the receiver. It is related with the distance between the nodes, the network congestion and the propagation velocity (a fraction of the light speed) among other parameters.

5.7.2 Platform description

The experiment presented in this report is completely carried out in *PlanetLab* , although the results obtained will be used to implement a realistic model of the networks in the communications between the simulators and the cloud system of the GEO-Cloud experiment implemented in *Virtual Wall* and *BonFIRE* respectively.

PlanetLab is a global research network that supports the development of new network services (Europe, 2014). This testbed currently consists of 1188 nodes at 582 sites, which allows researchers to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables and query processing. Currently it is split into two platforms: *PlanetLab Europe* that contains the European nodes and *PlanetLab Central* which contains the nodes located outside Europe. For the experiment we use nodes from both of them.

5.7.3 Tools description

To implement and execute the experiment we use the following tools:

- *NEPI* «(INRIA, 2014)»: It is a Python-based language library used to design and easily run network experiments on network evaluation platforms (e.g. PlanetLab, OMF wireless testbeds and network simulators among others). It facilitates the definition of the experiment workflow, the automatic deployment of the experiment, resource control and result collection; and has the functionalities of automatic provisioning of resources and automatic deployment of the experiment. In the experiment NEPI is used to provision the nodes and execute the whole experiment.
- *Iperf* «(Iperf, 2014)»: It is a tool used to measure the maximum *TCP* bandwidth, allowing the tuning of various parameters and *UDP* characteristics. *Iperf* reports *band-*

width, *delay jitter* and *datagram loss*. This software allows any host to play the client and server roles. In the experiment, it is used to obtain the bandwidth with a step of one second when executed in *TCP* mode and the loss rate when executed in *UDP* mode. The nodes in Layer 1 and 2 are configured as clients and the cloud node as server.

- *Ping* «(Pelsser, Cittadini, Vissicchio, Bush, 2013)»: It is software used to test if a host on an Internet Protocol is reachable. It measures the **RTT!** (**RTT!**) for messages sent from the originating host to a destination host. In the experiment, it is used to measure the latency delivery of a package over the Internet between the nodes in Layer 1 and 2 and the central node.

5.7.4 PlanetLab Experiment

The objective of the *GEO-Cloud* experiment is to simulate as realistically as possible the behaviour of a complete Earth Observation system. With this aim, the communication links in the real system have to be modelled to connect the simulators implemented in *Virtual Wall* and *BonFIRE* with the values obtained from the experiment in *PlanetLab* . The experiment then consists of communicating 12 real nodes representing the ground stations (the nearest *PlanetLab* node to the real ground station was selected) and the end users distributed around the world (we selected 31 nodes from different 31 countries) with a node representing the cloud (located in *INRIA*) to measure the real impairments of the networks and to implement a realistic model of the communications. The impairments to be measured and used to model the network are the effective bandwidth, the latency and the loss rate. An equivalence scheme is shown in Figure 5.25 with the correlation between the parameters obtained from the experiment and the inputs to model the links between *Virtual Wall* and *BonFIRE* . There are two networks in the system:

1. The dedicated network connecting the ground stations and the cloud: it is represented by the bandwidth, the latency and the loss rate.
   (a) The bandwidth will be computed as a control variable.
   (b) The latency will be extracted from the latency measured in the *PlanetLab* experiment.
   (c) The loss rate will be extracted from the loss rate measured in the PlanetLab experiment.
2. The Internet network connecting the end users and the cloud: it is represented by the bandwidth, the latency, the loss rate and the background traffic.
   (a) The bandwidth will be computed as a control variable.
   (b) The latency will be extracted from the latency measured in the *PlanetLab* experiment.
   (c) The loss rate will be extracted from the loss rate measured in the *PlanetLab* experiment.

(d) The background traffic is affected by the following parameters:

(e) Throughput: the effective bandwidth measured with the *PlanetLab* experiment will be computed as the throughput parameter in *Virtual Wall* .

3. Packet size: 1500 bytes.

4. Protocol: the protocol used is TCP.

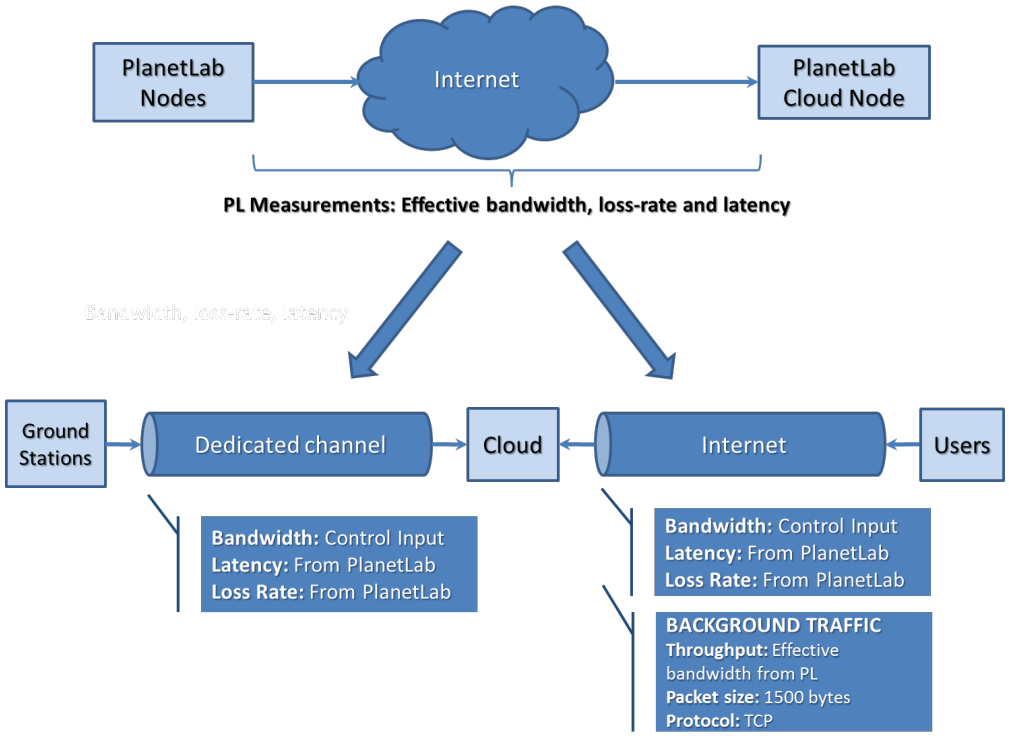


Figure 5.25: PlanetLab and Modelled Links Equivalences

5.7.4.1 System Modeling

The real system is modeled into three main components: i) a network of ground stations acquiring imagery data from a constellation of optical satellites, ii) a cloud infrastructure that ingests the data from the ground stations, processes it, stores it and distributes it through web services and iii) end users around the world accessing to the web services offered. The system can be divided into two layers:

1. *Layer 1* is constituted by 12 ground stations connecting with a cloud infrastructure. The ground stations and their location are depicted in Table 5.11. Their locations and footprints are depicted in Figure 5.26. The footprints represent the area in which the satellites can establish the communication with the ground stations.

2. *Layer 2* is constituted by the end users accessing the web services implemented in cloud. These users are distributed around the world and can be governments, emergency services, media and individuals among others.

Python Library	Function
Sys	System library
OS	Operative system interactions supply
Sched	Library that allow the Satellite Simulator to schedule the task along the time
Time	For managing the time
Socket	Library for creating and establishing connections with other host
Pdb	Used for debugging the software
Logging	Log

Table 5.10: Ground Station Simulator Python Libraries

Ground Station	Country of GS location
Irkutsk	Russia
Puertollano	Spain
Svalbard	Norway
Troll	Antarctic
Chetumal	Mexico
Córdoba	Argentina
Dubai	United Arab Emirates
Kourou	French Guiana
Krugersdorp	South Africa
Malaysia	Malaysia
Prince Albert	Canada

Table 5.11: Ground Station Location

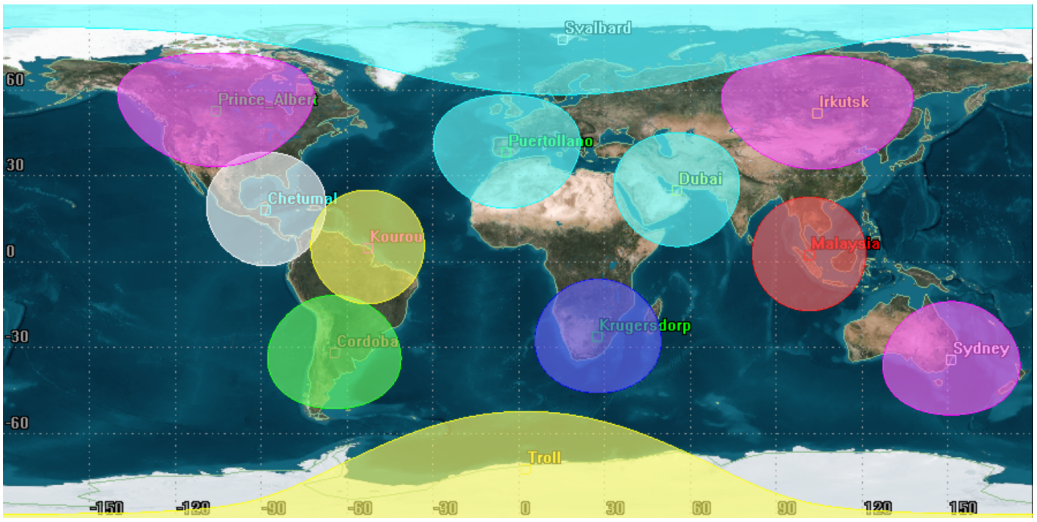


Figure 5.26: Footprints of the ground stations

From the previous layers two networks can be identified: the network between the ground stations and the cloud and the network between the end users and the cloud. The system and the interconnections between components are depicted in Figure 5.27. The connections between the ground stations and end users with the cloud are represented as arrows with different line types to represent that every connection can have different characteristics and impairments. All the connections are *TCP*.

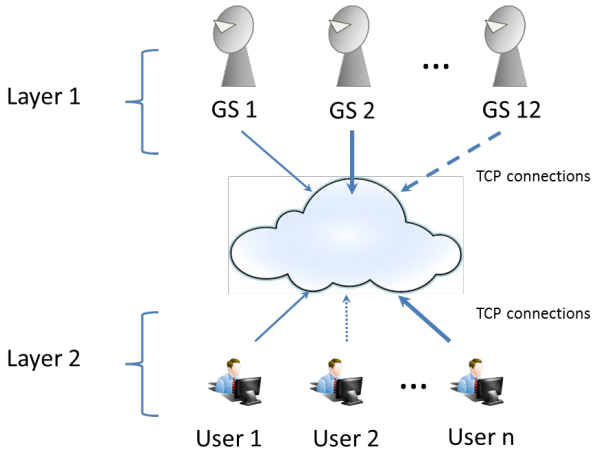


Figure 5.27: System Description

We define the network in function of the following representative impairments: *effective bandwidth, latency and loss rate*. Thus, every link is represented in function of the previous impairments: effective bandwidth, latency, loss rate. This system is implemented in *Virtual Wall* and *BonFIRE* as depicted in Figure 5.28. The experiment in *PlanetLab* will be used to update the network parameters connecting *Virtual Wall* and *BonFIRE* .

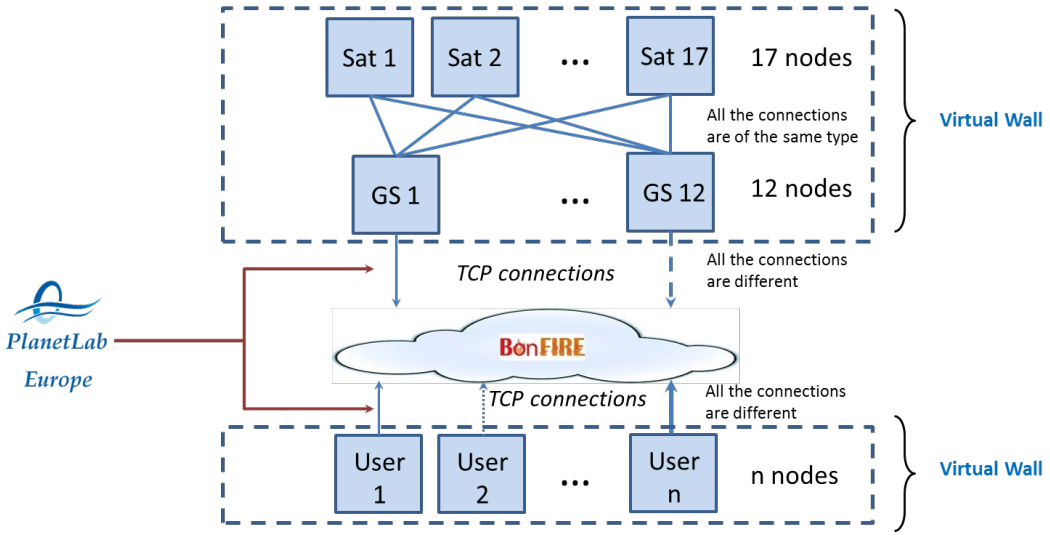


Figure 5.28: Scheme of the system implemented in Geo-Cloud

5.7.4.2 Network Design

The model of the ground stations network, the cloud and the end users accessing the web services provided was simplified to a set of interconnected nodes. The network was divided into two layers connected by a central node representing the cloud servers for similarity with the real system:

- **Layer 1:** it represents the connections between 12 nodes representing the ground stations and a central node representing the cloud servers. In *PlanetLab Europe* and *PlanetLab Central*, the nearest nodes to the real location of the ground stations were selected. For the central node, a node in *INRIA* was chosen, since the *BonFIRE* cloud has servers in the same location. This layer then represents the transfer of geodata acquired by the constellation of satellites from the ground stations in which the data is downloaded to the cloud. The network topology implemented is peer-to-peer, i.e. each node representing the ground stations is directly connected with the central node. In Table B.1 the *PlanetLab* selected nodes for layer 1 and for the cloud central node are shown. The nodes are numbered in ascending order in function of the distance to the central node, i.e. the closest node is the number 0 and the furthest the 37.

- **Layer 2:** it represents the connection between the central node representing the cloud servers and the end users. 31 different nodes were selected in *PlanetLab Europe* and *PlanetLab* Central in 31 different countries around the world. This allows us to have a representative sample of global users accessing the web service s. In this case, the network topology is also peer-to-peer. In Table Table B.2 the nodes selected for layer 2 are listed. We tried to increase the number of nodes in different countries, but during the execution of the experiment we did not find available PlanetLab nodes in the following countries: Austria, Cyprus, Denmark, Egypt, Ecuador, Iceland, India, Jor-

dan, Mexico, Pakistan, Puerto Rico, Romania, Slovenia, Sri Lanka, Tunisia, Turkey, Venezuela, Uruguay and Taiwan.

Figure 5.29 shows a scheme representing the network created in *PlanetLab* . The connections between the nodes are *TCP*.

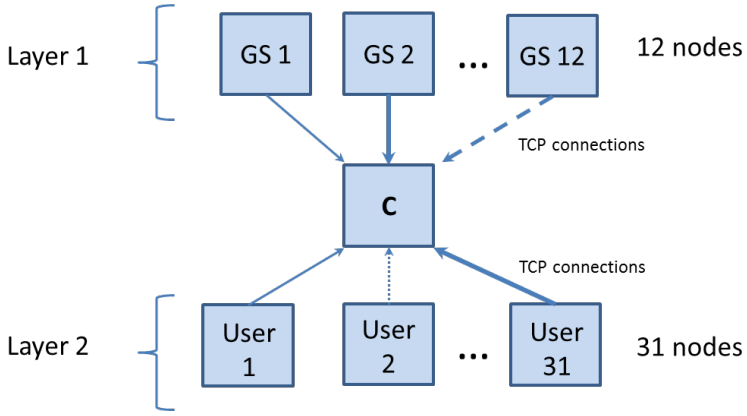


Figure 5.29: PlanetLab Network Scheme

5.7.4.3 Experiment Design and Execution

The experiment is designed to measure the impairments of the network. Those impairments are required parameters in *Virtual Wall* to deploy a topology network in such a testbed. Then, in this experiment we measured latency, loss-rate and effective bandwidth. The deployment of the experiment was done with *NEPI* , in which *Iperf* and *Ping* were implemented to measure the impairments. The experiment consists of establishing communications between any node in layer 1 or layer 2 with the central node and measuring the previously described impairments. 21600 trials will be done during 6 hours of the experiment execution in steps of one second for each pair of nodes, i.e. a node from layer 1 or 2 and the central node.

The software developed to measure the impairments is constituted of 6 scripts:

- Script to measure the effective bandwidth in the ground stations nodes: “bandwidthGS.py”.
- Script to measure the effective bandwidth in the end users nodes: “bandwidthEndUser.py”
- Script to measure the latency in the ground stations nodes: “latencyGS.py”.
- Script to measure the latency in the end users nodes: “latencyEndUser.py”
- Script to measure the loss rate in the ground stations nodes: “lossRateGS.py”
- Script to measure the loss rate in the end users nodes: “lossRateEndUser.py”

The pair of scripts that measure the same impairment are differentiated one from each other in the provisioning of the nodes. Those nodes representing the ground stations are

manually selected, while the end users nodes are automatically provisioned by *NEPI* by indicating the country name as parameter. This parameter allows *NEPI* to select an available node in that country.

The previous six scripts are individually executed in a local host and they start their workflow.

5.7.4.3.1 The bandwidthGS.py script

The bandwidthGS.py script measures the effective bandwidth in the ground stations nodes. When it is executed it carries out the next tasks:

1. Provisioning of the nodes that were manually selected.
2. Creation of the commands to be uploaded:
   - In the cloud node

     timeout %dm iperf -s -f m -i 1 -p %d

     Timeout is a command that executes a program during a specified time %dm in minutes. For this experiment dm was chosen to be 65 minutes.
     - s indicates that Iperf is executed in server mode
     - -f m indicates the format to report the received data. In this case in Mb.
     - i 1 Periodic reports every 1 second
     - p %d indicates the port to listen. In our case 20004.
   - In the ground station nodes *iperf -i 1 -f m -c %s -t %d -p %d -y c > node%d.out*
     - i 1 Periodic reports every 1 second.
     - f m indicates the format to report the received data. In this case in Mb.
     - c %s indicates the server to establish the communication with.
     - t indicates the data transmission time. In our case 3600 seconds.
     - p %d indicates the port to listen. In our case 20004.
     - y c> node%d.out indicates that the report format is csv. The output file is node%d.out, where %d indicates the number of the node tested.

   By default Iperf is executed in TCP mode.
3. Uploads the commands to the nodes
4. Executes the command in cloud
5. Executes the command in the rest of nodes
6. During the execution of the commands the data is collected
7. Finishes the execution of the commands

8. The data collected is retrieved
9. The resources are released.

The flow diagram of the effective bandwidth measurements in the ground station nodes is depicted in Figure 6 a.

5.7.4.3.2 The bandwidthEndUser.py script

The *bandwidthEndUser.py* script measures the effective bandwidth in the end users nodes. When it is executed it carries out the next tasks:

1. Automatic provisioning of the nodes.
2. Tasks 2 to 9 of the bandwidthGS.py script.

The flow diagram of the effective bandwidth measurements in the end users nodes is depicted in Figure 6 b.

5.7.4.3.3 The lossRateGS.py script

The lossRateGS.py script measures the loss rate in the ground stations nodes. When it is executed it carries out the next tasks:

1. Provisioning of the nodes that were manually selected.
2. Creation of the commands to be uploaded:
   - In the cloud node

     timeout %dm iperf -s -f m -i 1 -p %d -u

     Timeout is a command that executes a program during a specified time %dm in minutes. For this experiment dm was chosen to be 65 minutes.
     - s indicates that Iperf is executed in server mode
     - -f m indicates the format to report the received data. In this case in Mb.
     - i 1 Periodic reports every 1 second
     - p %d indicates the port to listen. In our case 20004.
     - u indicates that the *Iperf* software is executed in *UDP* mode.
   - In the ground station nodes *iperf -i 1 -f m -c %s -t %d -p %d -y c > node%d.out*

     - i 1 Periodic reports every 1 second.
     - f m indicates the format to report the received data. In this case in *Mb*.
     - c %s indicates the server to establish the communication with.
     - t indicates the data transmission time. In our case 3600 seconds.

- p %d indicates the port to listen. In our case 20004.
- y c> node%d.out indicates that the report format is csv. The output file is node%d.out, where %d indicates the number of the node tested.
- u indicates that the *Iperf* software is executed in *UDP* mode.

By default Iperf is executed in TCP mode.

3. Uploads the commands to the nodes
4. Executes the command in cloud
5. Executes the command in the rest of nodes
6. During the execution of the commands the data is collected
7. Finishes the execution of the commands
8. The data collected is retrieved
9. The resources are released.

The flow diagram of the loss rate measurements in the ground station nodes is depicted in Figure 6 a.

5.7.4.3.4 The lossRateEndUser.py script

The lossRateEndUser.py script measures the loss rate in the end users nodes. When it is executed it carries out the next tasks:

1. Automatic provisioning of the nodes.
2. Tasks 2 to 9 of the lossRateGS.py script.

The flow diagram of the loss rate measurements in the end users nodes is depicted in Figure 6 b.

5.7.4.3.5 The latencyGS.py script

The latencyGS.py script measures the latency in the ground stations nodes. When it is executed it carries out the next tasks:

1. Provisioning of the nodes that were manually selected.
2. Creation of the commands to be uploaded: *ping %s -w %d*
   - %s indicates the host to do ping
   - w %d indicates the time of the ping execution.
3. Uploads the commands to the nodes

4. Executes the command in cloud
5. Executes the command in the rest of nodes
6. During the execution of the commands the data is collected
7. Finishes the execution of the commands
8. The data collected is retrieved
9. The resources are released.

The flow diagram of the latency measurements in the ground station nodes is depicted in Figure 7 a.

5.7.4.3.6 The latencyEndUser.py script

The latencyEndUser.py script measures the loss rate in the end users nodes. When it is executed it carries out the next tasks:

1. Automatic provisioning of the nodes.
2. Tasks 2 to 9 of the latencyGS.py script.

The flow diagram of the latency measurements in the end users nodes is depicted in Figure 7 b.

5.7.5 Conclusions

In this document the part of the GEO-Cloud experiment executed in PlanetLab is presented. Topology networks have been created in two layers to simulate the communications between the ground stations and the cloud infrastructure and between such a cloud infrastructure with end users around the world accessing web services computed in cloud. The experiment consists of measuring the real network impairments (effective bandwidth, latency and loss rate) to obtain an approach for its later implementation in the complete Earth Observation system simulator implemented in Virtual Wall and BonFIRE.

5.8 GEO-Cloud GUI

In this section, the GUI developed for managing the execution of the defined scenarios in GEO-Cloud experiment is exposed. First, the architecture is explained. Then, the components as *ExperimentController*, *Video Component* or *Load Plot* among others are described. Finally, the GUI is shown and an example execution step by step is performed.

5.8.1 Architecture

The graphical user interface of GEO-Cloud experiment is composed by some components:

- *Experiment Controller:*
- *RemoteConnection:*

5.8.1.1 Experiment Controller

5.8.1.2 RemoteConnection Class

5.8.2 Execution

Chapter 6

Project Evolution and Cost

6.1 Project Evolution

6.2 Cost

Chapter 7

Results

In this chapter, the obtained results of the implementation, execution and validation of the GEO-Cloud experiment are explained. The *PlanetLab* experiment for measuring the network impairments (bandwidth, loss-rate and **RTT!**) are shown and plotted. These plots represents how the impairments directly depends on the distance between both source and destination elements. The next section pictures the results of the execution of defined scenarios into the GEO-Cloud experiment. Finally, the evaluation of the implementation for EO images processing on-cloud is explained.

7.1 PlanetLab Experiment Results

7.2 GEO-Cloud Experiment Results

Chapter 8

Conclusions

This file contains the conclusionso

APPENDIX

Appendix A

User Manual

A.1 First steps

Appendix B

PlanetLab Nodes

Ground Station	Node Location	Selected Node	Site	Node number
Irkutsk[1]	China	planetlab1.buaa.edu.cn	Beihang University	24
Puertollano	Spain	Planetlab2.dit.upm.es	Universidad Politécnica Madrid	5
Svalbard	Norway	planetlab1.cs.uit.no	University of Tromso	14
Troll[2]	New Zealand	planetlab1.cs.otago.ac.nz	University of Otago	37
Chetumal[3]	USA	Planetlab1.eecs.ucf.edu	University of Central Florida	22
Córdoba	Argentina	planet-lab2.itba.edu.ar	Instituto Tecnológico Buenos Aires	34
Dubai[4]	Israel	planet1.cs.huji.ac.il	The Hebrew University of Jerusalem	19
Kourou[5]	Brazil	planetlab1.pop-pa.rnp.br	RNP	26
Krugersdorp[6]	Reunion Island (France)	lim-planetlab-1.univ-reunion.fr	Universite de La Reunion	28
Malaysia	Malaysia	planetlab1.comp.nus.edu.sg	National University of Singapore	32
Prince Albert	Canada	planetlab-2.usask.ca	University of Saskatchewan	21
Sidney	Australia	pl1.eng.monash.edu.au	National ICT Australia	36
Cloud[7]	France	ple6.ipv6.lip6.fr	University Pierre et Marie Curie	N/A

Table B.1: Ground Segment Nodes

Country	Selected Node	Number	Country	Selected Node	Number
Argentina	planet-lab2.uba.ar	35	Japan	planet1.pnl.nitech.ac.jp	31
Australia	pl1.eng.monash.edu.au	36	Korea, Republic of	netapp7.cs.kookmin.ac.kr	27
Belgium	rochefort.infonet.fundp.ac.be	2	The Netherlands	planetlab1.cs.vu.nl	4
Brazil	planetlab1.pop-pa.rnp.br	26	New Zealand	planetlab1.cs.otago.ac.nz	37
Canada	planetlab-2.usask.ca	21	Norway	planetlab1.cs.uit.no	14
China	planetlab1.cqupt.edu.cn	25	Poland	ple2.dmcs.p.lodz.pl	13
Czech Republic	planetlab1.cesnet.cz	8	Portugal	planet1.servers.ua.pt	11
Finland	planetlab-1.research.netlab.hut.fi	17	Russian Federation	plab1.cs.msu.ru	20
France	inriarennes2.irisa.fr	0	Singapore	planetlab1.comp.nus.edu.sg	33
Germany	planetlab02.tkn.tu-berlin.de	6	Spain	dplanet2.uoc.edu	3
Greece	planetlab1.ionio.gr	15	Sweden	planetlab2.s3.kth.se	16
Hong Kong	planetlab1.ie.cuhk.edu.hk	30	Switzerland	planetlab2.unineuchatel.ch	1
Hungary	planet2.elte.hu	12	Thailand	ple2.ait.ac.th	29
Ireland	planetlab-node-01.ucd.ie	10	United Kingdom	planetlab-2.imperial.ac.uk	9
Israel	planetlab2.tau.ac.il	18	United States	planetlab-04.cs.princeton.edu	23
Italy	planet-lab-node1.netgroup.uniroma2.it	7			

Table B.2: User Nodes

Bibliography

[Mar08] J. Martínez de Sousa. *Ortografía y ortotipografía del español actual*. Trea, 2008.

This document was edited and typed with LaTeX
by using the **arco-pfc** template whose is available in the following address:
`https://bitbucket.org/arco_group/arco-pfc`