|  |  |  |  |
| --- | --- | --- | --- |
| logofire-standardresolution | jaune | logo_ce-en-rvb-hr | 7plogo |

|  |  |
| --- | --- |
| Project Acronym | **Fed4FIRE** |
| Project Title | **Federation for FIRE** |
| Instrument | **Large scale integrating project (IP)** |
| Call identifier | **FP7-ICT-2011-8** |
| Project number | **318389** |
| Project website | **www.fed4fire.eu** |
| Experiment | **GEO-Cloud** |

**Orchestrator and Archive & Catalogue Software Document**

|  |  |
| --- | --- |
| Work package | WP10 |
| Task | T10.1.2 GEO-Cloud Experiment |
| Due date | 28/02/2014 |
| Submission date | 28/02/2014 |
| Report Lead | Félix Pedrera (DEIMOS) |
| Version | 1.0 |
| Authors | Félix Pedrera, Manuel José Latorre, Rubén Pérez (DEIMOS) |
| Reviewers | Jonathan Becedas (DEIMOS) |

|  |  |
| --- | --- |
| **Document ID** | **GEO-Cloud-DMS-TEC-TEC09-E** |
| Abstract | This document provides the detail design of Orchestrator component. |
| Keywords | Report |

|  |  |  |  |
| --- | --- | --- | --- |
| Nature of the document | R | Report | X |
| P | Prototype |  |
| D | Demonstrator |  |
| O | Other |  |
| Dissemination level | PU | Public |  |
| PP | Restricted to other programme participants (including the Commission) |  |
| RE | Restricted to a group specified by the consortium (including the Commision) |  |
| CO | Confidential, only for members of the consortium (including the Commission) | X |

**Disclaimer**

*The information, documentation and figures available in this deliverable, is written by the Fed4FIRE* (Federation for FIRE) *– project consortium under EC co-financing contract FP7-ICT-318389 and 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.*

**Executive Summary**

In this document the design of the orchestrator is presented. It manages the product processors, the archive and catalogue and the cloud resources. It is also responsible of the images ingestion in the cloud from the ground stations implemented in Virtual Wall.

Furthermore, the archive and catalogue module is designed by using GeoServer. It provides a CSW interface with which the Image Distribution and Visualization (IDV) system communicates to provide the web service to the end users.

The interfaces of the orchestrator and the archive and catalogue module are also presented.

**Acronyms and Abbreviations**

|  |  |
| --- | --- |
| F4F | Fed4FIRE |
| XML | eXtensible Markup Language |
| GS | Ground Station |
| CSW | Web Catalogue Service |

**Table of Contents**

[1 Introduction 7](#_Toc382378080)

[2 Design of the orchestrator 8](#_Toc382378081)

[2.1 Functionality of the orchestrator 8](#_Toc382378082)

[2.2 Interactions of the Orchestrator with the other modules of the GEO-Cloud architecture 8](#_Toc382378083)

[2.3 Orchestrator’s working flow 9](#_Toc382378084)

[2.4 Interfaces of the Orchestrator 10](#_Toc382378085)

[2.4.1 Interfaces with the Ground Stations implemented in Virtual Wall 10](#_Toc382378086)

[2.4.2 Interfaces with the Product Processors 11](#_Toc382378087)

[2.4.3 Interfaces with the Archive and Catalogue 12](#_Toc382378088)

[1 Detailed Design of the orchestrator 14](#_Toc382378089)

[1.1 Use Cases 14](#_Toc382378090)

[1.2 Architecture of the Orchestrator 15](#_Toc382378091)

[1.3 Flow Diagrams 17](#_Toc382378092)

[1.3.1 Pooling GS 17](#_Toc382378093)

[1.3.2 Get Raw Data 18](#_Toc382378094)

[1.3.3 Generate Job Order 19](#_Toc382378095)

[1.3.4 Create Processing Chain 20](#_Toc382378096)

[1.3.5 Image Processing 21](#_Toc382378097)

[1.3.6 Delete Processing Chain 22](#_Toc382378098)

[2 Software developed 23](#_Toc382378099)

[3 Conclusions 24](#_Toc382378100)

[3 Bibliography 25](#_Toc382378101)

# Introduction

This document describes the design of the Orchestrator. The Orchestrator is the component that manages the tasks to be done in the cloud. The orchestrator will be running over the BonFIRE Cloud and it will control all interactions between all the components implemented in Fed4FIRE test beds (Fed4FIRE, 2014) as BonFIRE or VirtualWall.

This document is divided into the following sections: the first section is devoted to the description of the Orchestrator’s functionalities and the interactions with the other components of the Geo-Cloud system; Section 2 describes the workflow of the Orchestrator and the interfaces with Geo-Server and the product processors; finally, Section 3 shows the detailed design of the Orchestrator.

# Design of the orchestrator

The Orchestrator manages the automatic distribution of the raw data to the processors, supports the Archive and Catalogue module and controls the cloud resources.

## Functionality of the orchestrator

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 they 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. Finally they are XML files.
* 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.

## Interactions of the Orchestrator with the other modules of the GEO-Cloud architecture

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

* Ground stations implemented in Virtual Wall
* Product Processors
* Processing instances in the cloud
* Archive and catalogue

Figure 1 depicts the Orchestrator’s interactions with the other modules of the GEO-Cloud architecture.

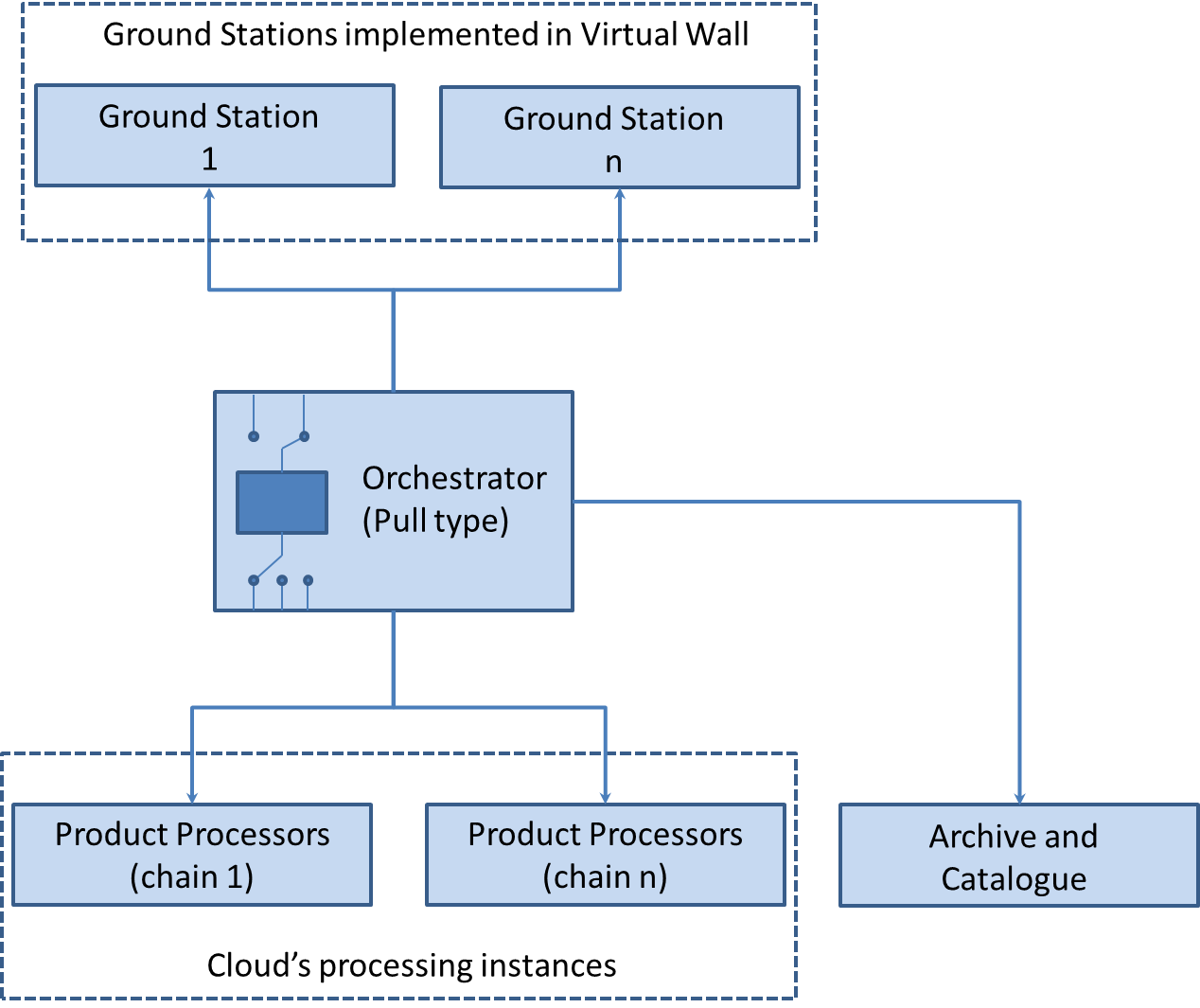


Figure Orchestrator interactions

As shown in Figure 1, 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.

## Orchestrator’s working flow

The Orchestrator works by following the next sequence of steps:

**Step 1:** The Orchestrator gets all the information about the Ground Stations and localizes them.

**Step 2:** The Listener pools to the Ground Stations and when there are a downloadable RawData, the New\_Data\_Event is launched.

**Step 2:** When the New\_Data\_Event occurs, the Orchestrator downloads the data.

**Step 3:** The Orchestrator moves the raw data to a shared storage.

**Step 4:** 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.

**Step 5:** The Orchestrator gets the ProcessorChainController object (this object was made regarding Singleton pattern).

**Step 6:** The Orchestrator instructs the ProcessorChainController object to create a new processing chain by sending the JobOrders created in step 4.

**Step 7:** The ProcessorChain Controller object creates a new Processing Chain to process the data.

**Step 8:** The Processing Chain sequentially executes the L0, L1A, L1B, L1C processors.

**Step 9:** When the ProcessingChain has finished, this notifies the ProcessorChainController object that the processing ended.

**Step 10:** The ProcessingChainController alerts the Orchestrator that the Processing Chain has finished.

**Step 11:** The Orchestrator takes the created image and puts it into the Archive.

Figure 2 depicts the workflow of the Orchestrator.

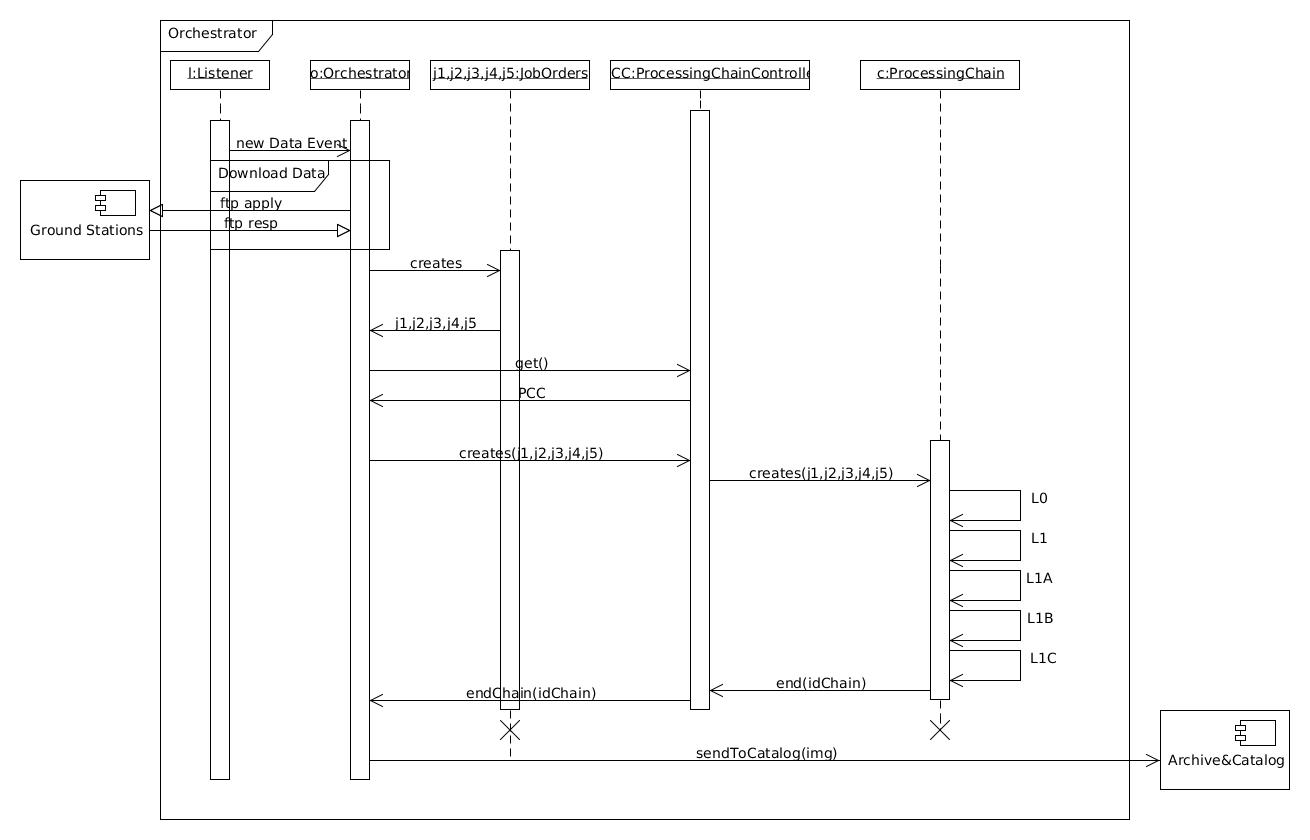


Figure Orchestrator workflow

## Interfaces of the Orchestrator

The Orchestrator has interfaces with the Ground Stations implemented in Virtual Wall, with the Product Processors and with the Archive and Catalogue.

### Interfaces with the Ground Stations implemented in Virtual Wall

The Ground Stations are deployed in some Virtual Wall nodes. In those, the impairments and features of the network are simulated. Essentially, the Orchestrator is pooling those Ground Stations over FTP connections to know when a raw data is available. So, this Ground Stations are FTP servers in which the Orchestrator can get the raw data obtained by the constellation of satellites.

### Interfaces with the Product Processors

The Orchestrator communicates with the Product Processors through the ProcessingChainController instance as shown in Figure 2. The Orchestrator commands the ProcessingChainController to create a new processing chain to process the raw data. When this process finishes, the ProcessingChainController sends to the Orchestrator a message to indicate the end of the chain. Thus, the ProcessingChainController checks the product processors progress and initiates the next level until the processing chain finishes. Finally, the Orchestrator obtains the end product and locates it in the Catalogue Service.

#### ProcessingChainController

As previously described, the ProcessingChainController interchanges messages with the Orchestrator to initialize the processing chain and obtain the processed data (image). Furthermore, this component controls all the levels of the Product Processors, i.e. L0, L1A, L1B and L1C.

The ProcessingChainController runs the Product Processors and sends to Orchestrator the final result, i.e. the processed image.

In the following sections the different processors are explained.

#### 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, i.e. it is constant for all the L1 levels.

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| The Raw Data | The L0 products |
| The configuration files |
| The JobOrder |

#### The L1A processor

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

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

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| The L0 scene | The L1A products |
| The configuration files |
| The JobOrder |

#### The L1B processor

Level 1B implements the geolocation, re-sampling and packing.

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| The L1A scene | The L1B products |
| The configuration files |
| The JobOrder |

#### The L1C processor

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

|  |  |
| --- | --- |
| **Inputs** | **Outputs** |
| The L1B product | The final image |
| The configuration files |
| The JobOrder |

### Interfaces with the Archive and Catalogue

#### Archive

The Archive is the storage module for the geodata generated from the Product Processors. This component is implemented by using the Geo-Server software (Geoserver, 2014). The Geo-Server software is explained in the GEO-CLOUD-DMS-TEC-TEC09-10 report.

The Orchestrator communicates with Geo-Server through the Gsconfig0.6.7 API (Winslow & Genthall, 2014) . Thus, when the Product Processors chain generates an image, the orchestrator transfers it to the Geo-Server's storage unit. Geo-Server is also notified by the ingestion of a new image in the archive the orchestrator. The Orchestrator also requests Geo-Server to create a new layer with the processed image and push it into its database. Then the meta-data of the image is uploaded to the Archive and a CSW interface is provided to the Image Distribution and Visualization module for accessing the Archive. This is schematically represented in Figure 3.

#### Catalogue

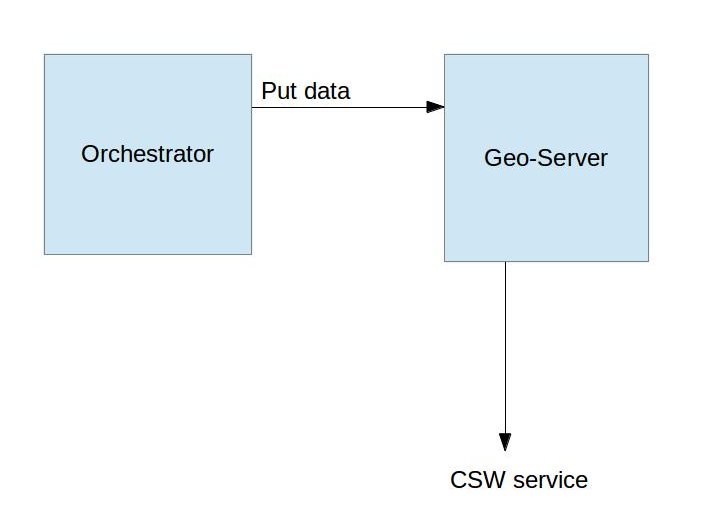
The Catalogue module offers to the end users all the images stored in the Archive. This service is implemented through the CSW standard. GeoServer has a plugin to manage the requests and accesses to the catalogue.

Figure Scheme of the communications between the orchestrator and Geo-Server

# Detailed Design of the orchestrator

In this section, the design of the Orchestrator is described. First, the use cases are shown for understanding the main task of the Orchestrator. Then, the diagram of the Orchestrator’s Architecture is introduced.

## Use Cases

The use case of the Orchestrator consists of actions that the different components can do. Figure 4 shows the Orchestrator´s use cases. There are several actions: “Create Processing Chain” or “Delete Processing Chain” and their interactions with actors like Archive&Catalogue or the Shared Storage of the Orchestrator.

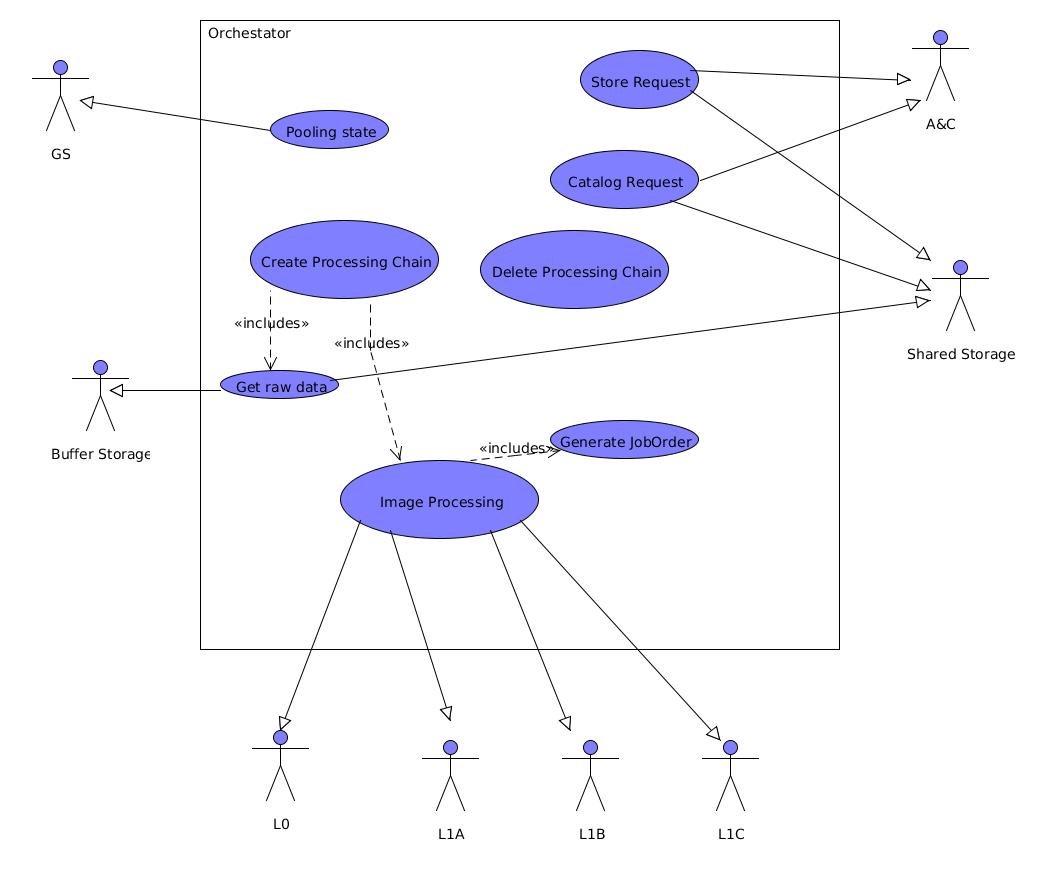


Figure Use Case of Orchestrator

In Table 1 the use cases are described.

Table Use cases description

|  |  |
| --- | --- |
| **Use Case** | **Explanation** |
| Pooling State | This state will pool the GroundStations frequently whether raw data is available. |
| Create Processing Chain | This use of case creates a new Processing Chain. Includes the Get Raw data use case and Image. Processing use case. |
| Image Processing | This use case essentially makes all procedure for processing the data (Executes sequentially the processors). |
| Generate Job Order | This case is included inside the Image Processing use case. This creates the necessary JobOrders for each Product Processor. |
| Delete Processing Chain | This use case removes the ended Processing Chain only. |
| Catalog Request | This use case communicates with the Catalog service for uploading the processing data (For future expansion). |
| Archive Request | This use case sends the processed data to Archive Component. |

## Architecture of the Orchestrator

The Architecture of the Orchestrator consists of three layers: Communications, Storage&Catalogue and Domain. The layers composing the Orchestrator are:

* **The Communications layer:** Contains the Listener. The Listener does the pooling to the Ground Stations and communicates with Orchestrator component.
* **The Storage&Catalog layer:** Contains the interfaces with the Storage and Catalogue. It uses the library GSConfig to communicate with GeoServer.
* **The Domain layer:** Contains all the logic and the functionality of the Orchestrator. There are some components in this packet, which are Orchestrator, ProcessingChainController and Chain Processing. Also the ProcessingChainController aggregates the JobOrders to be able to create the Processing Chain. The interactions between this components is explained in section 2.3

Table 1 shows all the components of the Architecture of the Orchestrator. In Figure 5 a scheme of the architecture is depicted.

Table Components of the Architecture of the Orchestrator

|  |  |  |
| --- | --- | --- |
| **Layer** | **Component** | **Description** |
| Communications | Listener | Realize pooling to GroundStations and throw GSReady signal. |
| Storage&Catalog | Proxy Catalog | Implements the ICatalog interface and provide to Orchestrator sending data to Catalog. |
| Proxy Storage | Implements the Storage interface |
| Domain | Orchestrator | Implements IOrchestrator and manage all system. |
| ChainProcessorController | It creates the processing chains and warns to Orchestrator when the chain is ended. |
| Chain Processing | Execute all processors sequentially. |
| JobOrder | It´s created, one for each processor, as input of processors. |

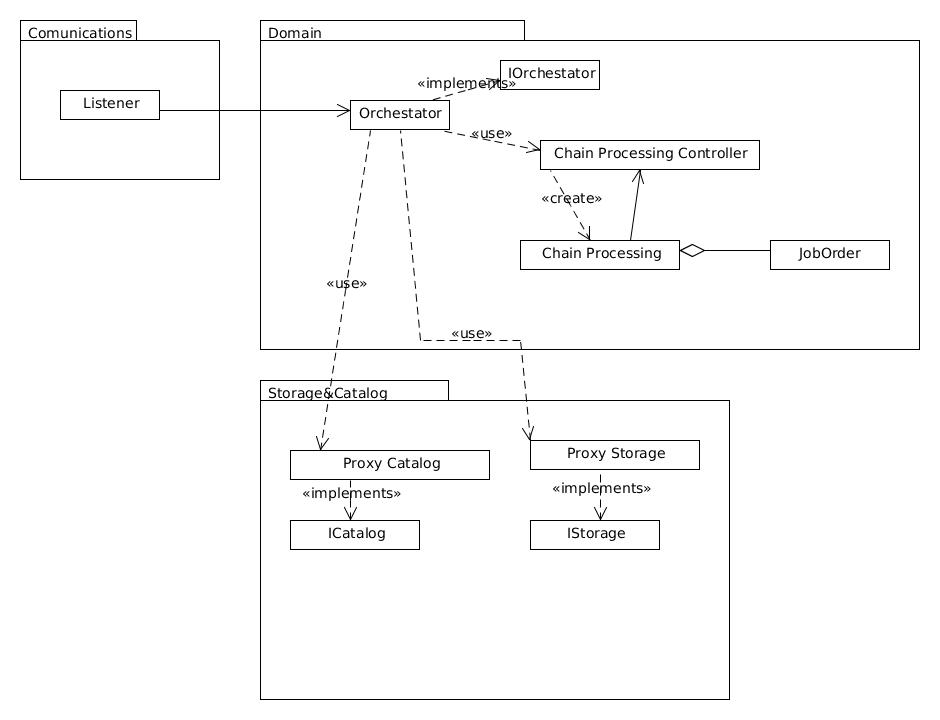


Figure Architecture of the Orchestrator

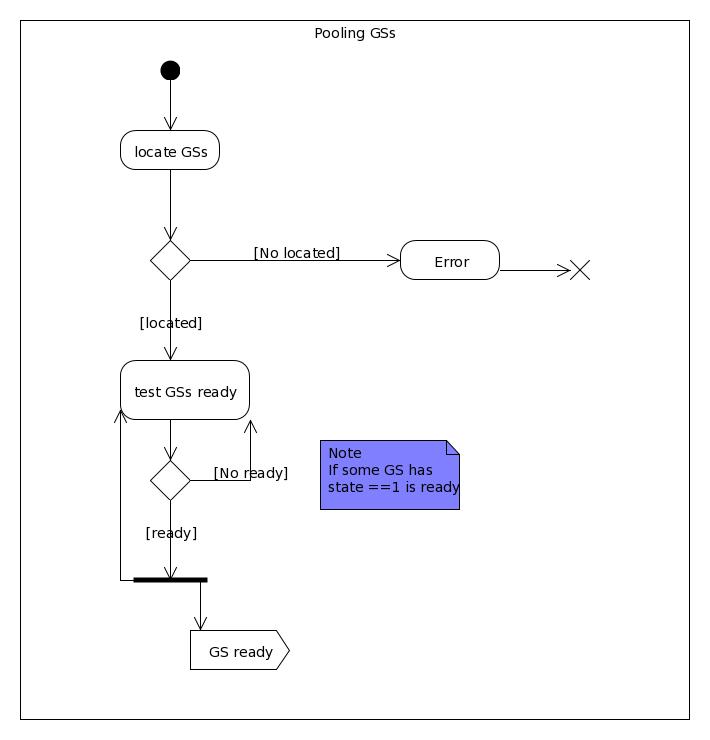
## Flow Diagrams

The following diagrams provide a low-level vision of each use case of the Orchestrator.

## Pooling GS

This case is executed by the Listener. It localizes the ground station and does the pooling over an ftp that communicates the GS with the cloud. If an error occurs while pooling it is notified, else, the listener establishes the communication with the GS. Then the listener looks at the ftp continuously. This is done in time steps of variable time, for example, 30 seconds. If there is raw data ready for downloading in the GS the listener gets the data and sends a signal of GS ready to the Orchestrator. And the Pooling GSs comes to the GSs ready state again to go on looking at the ftp port for new data to be downloaded. The availability is defined with respect to the name of the data file. The Ground Station when has finished to download from some satellite, changes the name of file to the following format: “W\_GS<NUM>\_<DATE>\_<SCENE>” where W means that the file is available, NUM indicates which Ground Station is, DATE indicates the hour in which the data was created and SCENE shows the Scenario belonging to.

Figure Pooling GSs flow diagram



## Get Raw Data

When above occurs, the Orchestrator catch the signal, and open a FTP connection within the Ground Station. Then, gets the Raw Data and puts into the Shared Storage for it can be accessible by processors. At this point, the Orchestrator is able to create a new Processing Chain to process the data just get.

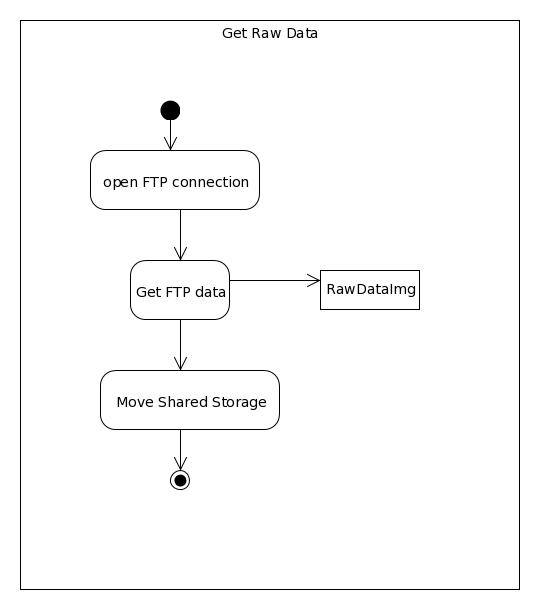


Figure Get Raw Data

## Generate Job Order

When a processing chain is created, also is necessary to build the JobOrders for each processor.

Each JobOrder contains the working directory of processor, directory of output data and temporally directory. As result of this, a JobOrder is generated with enough information. A JobOrder must be created for each level of processing.

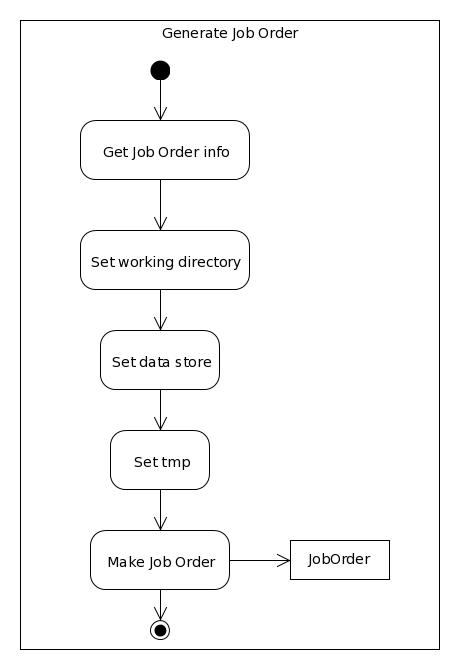


Figure Create JobOrder

## Create Processing Chain

When the Orchestrator catch the GSReady signal, obtains the useful Raw Data to process (section 1.3.2) and creates and sends the necessary information to bulid the JobOrders to ProcessingChainController. Then, the ProcessingChainController creates a new thread for executing a new Processor Chain and the necessary JobOrders. Finally, the processing of the image is made (section 1.3.5).

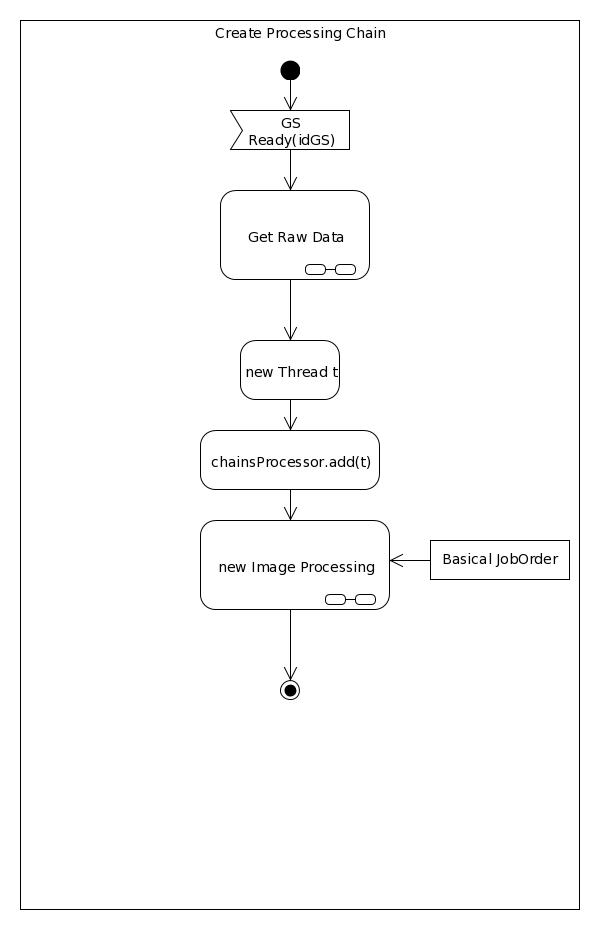
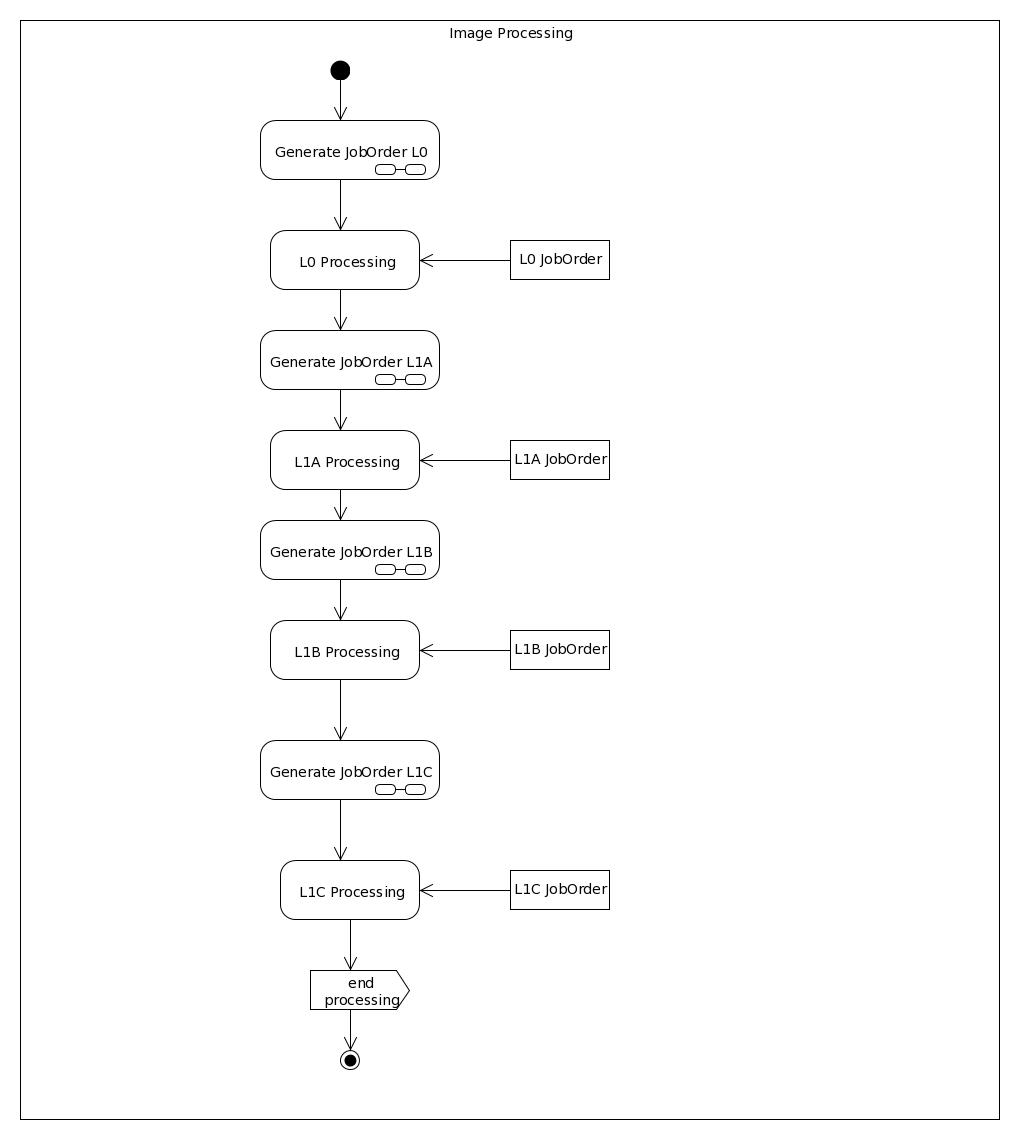


Figure Create Processing Chain

## Image Processing

This procedure is summery in two main parts. First, the generation of each JobOrder for corresponding processor. The basically information has arrived and the necessary JobOrders must be built (section 1.3.3). Then, the respective processing by processors. First, the L0 processing is made with corresponding JobOrder. Then the L1A processing and the other successively. At the end, end processing signal is thrown to ProcessorChainController.

Figure Image Processing



## Delete Processing Chain

When the ProcessorChainController has received the end Processing signal, the chain is deleted. The resources are free in order to create another processing chain in other time.

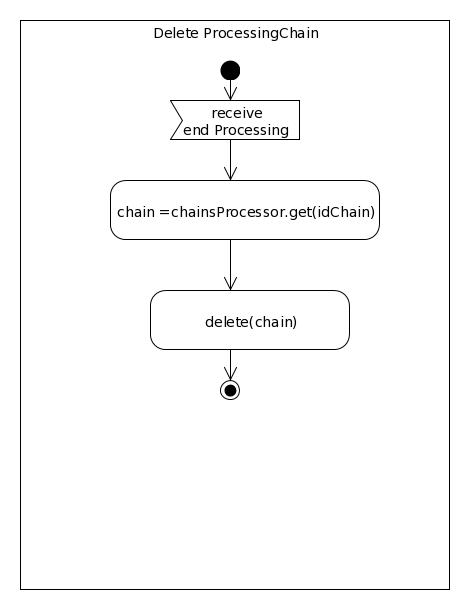


Figure Delete Processing Chain

# Software developed

The software that has been developed is indicated in Table 2. It has been developed in Python2.7 (Foundation, 2014).

Table Developed software for the Orchestrator

|  |  |
| --- | --- |
| **File** | **Contains** |
| jobOrder.py | Definition of JobOrder class |
| Listener.py | Contains the listener definition |
| Orchestrator.py | Contains the Orchestrator definition |
| ProcessingChain.py | Contains the Processing Chain Controller and the processing chain class definitions |
| Main.py | File than runs whole Orchestrator |

# Conclusions

The Orchestrator component and its subcomponents have been designed and developed. In addition, the GeoServer interface has been implemented.

For the GeoCloud experiment, this is a first approach and some parts can be adapted during the implementation

Table Summary of the interfaces of the Orchestrator

|  |  |  |
| --- | --- | --- |
| **From** | **To** | **Interface** |
| Listener | Orchestrator | Notify that Raw Data is available. |
| Orchestrator | ProcessorChainController | Orders to create a new processing chain and manages it. |
| ProcessorChainController | Orchestrator | Notify that a processing chain has finished. |
| Ground Stations | Listener | Pooling for available Raw Data. |
| Orchestrator | Archive | Through GSConfig library, to notify GeoServer that a new image has been putted. |
| Archive | Catalog | Through plugin CSW that supports catalog services. |

# Bibliography

Fed4FIRE. (2014). Fed4FIRE: Tools. *Fed4FIRE: Tools*.

Foundation, P. S. (2014, March). Python v2.7.6 documentation. *Python v2.7.6 documentation*.

Geoserver. (2014, March). Geoserver. *Geoserver*.

Winslow, D., & Genthall, S. (2014, March). GSConfig0.6.7. *GSConfig0.6.7*.