ETL for the integration of remote sensing data

Romero Jure, P. V.[1,2][0000−0001−5892−2137] paula.romero@mi.unc.edu.ar,
Cabral, J. B.[1,3][0000−0002−7351−0680] jbcabral@unc.edu.ar, and Masuelli,
S.[2] smasuelli@unc.edu.ar

[1] Gerencia de Vinculación Tecnológica, Centro Espacial Teófilo Tabanera, CONAE sede Córdoba, Argentina
[2] Facultad de Matemática, Astronomía, Física y Computación; Universidad Nacional de Córdoba, Córdoba, Argentina
[3] Instituto de Astronomía Teórica y Experimental, Observatorio Astronómico de Córdoba, Córdoba, Argentina

Abstract. Modern in-orbit satellites and other available remote sensing tools have generated a huge availability of public data waiting to be exploited in different formats hosted on different servers. In this context, ETL formalism becomes relevant for the integration and analysis of the combined information from all these sources. Throughout this work, we present the theoretical and practical foundations to build a modular analysis infrastructure that allows the creation of ETLs to download, transform and integrate data coming from different instruments in different formats. Part of this work is already implemented in a Python library which is intended to be integrated into already available workflow management tools based on acyclic-directed graphs which also have different adapters to impact the combined data in different warehouses.

Keywords: ETL · Satellite Imagery · Data Processing.

1 Introduction

The Extraction, Transformation, and Loading (ETL), is the formalism for extracting data from various sources, transforming it into a useful format, and loading it into a target repository, such as a data warehouse.

The term gained popularity throughout the industry around the 1970s rather than being formally defined in a document. However, previous works have settled the bases for the formalism. One of the first works is [6], which widely describes the process and its relation with Data Warehouse. Furthermore, [22] defines ETL activities and provides formal foundations for its conceptual representation.

ETL serves as a practical theoretical framework for data integration by simplifying the extraction of data from different sources, their transformation into a consistent and compatible format, and their loading into a centralized data warehouse to facilitate subsequent analysis.

In this context, remote sensing instruments on board artificial satellites orbiting the Earth are generating huge amounts of data every day, which is considered to be a Big Data problem [19] [5]. The data is transmitted to Earth, stored in a data warehouse system and usually provided to the user in some scientific file format, such as a Network Common Data Form (NetCDF) [15], Hierichal Data Format [9] or GeoTIFF[7]. The database where the files are stored and the format depends on the agency or organization responsible for each satellite. There exist several situations where someone needs to analyze Earth Observation data by combining measurements from multiple instruments onboard different satellites. That would be an appropriate problem for ETL formalism because we would be dealing with big data stored in different sources and we need to transform it into a product and load the latter in some database.

An event where different (satellite) sensors observe the same location roughly at the same time is called a collocation [12]. Several works have required implementing the collocation finding procedure, for example, [23] generated a dataset with combined data from MODIS and the Cloud Profiling Radar (CPR) on-board CloudSat, to study cloud types. [12] have studied collocations between the Microwave Humidity Sounder (MHS) on-board NOAA-18 and the CPR. In practice, collocating Earth Observation data usually comes with many problems due to the different sources where the data is stored and the compatibility of data formats.

In this context, we have decided to use the ETL formalism to integrate remote sensing data from multiple sources by designing extractors, transformers and loaders that access information from platforms provided by different missions. Although there are precedents of application [19], in our work we opted for a modular mechanism so that different users can customize their processing and analysis pipelines.

Although there are several programs suitable for performing collocations, particularly Geographic Information System (GIS) software, we have decided to implement all this infrastructure in Python, given its popularity and ecosystem in scientific computing [13]. Besides, even though there exist some Python libraries that implement data processing methods for Earth Observation data such as *Satpy* [8], we have not found any that implement extraction methods and integrate them with the processing stage, most of them assume that the data is available in the local system.

This paper is organized as follows: In Section 2 we present the ETL formalism and its relation to remote sensing data, then. In Section 3 we present and explain our design of a general ETL modular pipeline intended to combine data from instruments on board different artificial satellites in orbit around the Earth. In Section 3 we present an implementation of the design as a Python package. In Section 4 we present the results in the former and in Section 5, conclusions and future work to be done.

2 ETL formalism

The acronym ETL (Extract, Transform, Load) emerged in the context of data warehousing around the 1970s. And it comprises the following stages: Extracting data from the original sources, quality assuring and cleaning data, conforming the labels and measures in the data to achieve consistency across the original sources, and delivering data in a physical form [6].

This stages can be represented generally as a kind of diagram. For example [22] have developed a graphical notation useful to "capture the semantics of the entities involved in the ETL process". We have adopted this notation to represent the design proposed. In the following paragraph, we will review some basic concepts that are needed to explain our proposed design.

Figure 1 shows all the elements that could be present in the diagram:

The *Attributes*, which are the minimum unit of information, represented with an oval shape. The *Concepts*, squares, are the entities of the source databases and are defined by a name and a finite set of attributes. The hexagons represent the *Transformations*, which are the parts of code that execute a task. Next, the *ETL_Constrains* are a finite set of attributes, on which the constraint is imposed and a single transformation that implements the enforcement of the constraint. Finally, the *Notes* contains comments.

It is important to note that all notation is UML[4] based but some forms do not have the same meaning.

In this work, we will focus on transformation operations that transform the input data.

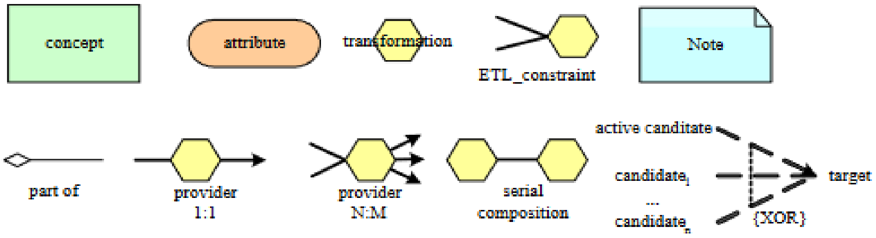


Fig. 1: Graphical notation for explaining an ETL pipeline. Figure courtesy of [22].

3 Design

Our design is based on [22] and is graphically represented in Figure 2a. For simplicity we have restricted our analysis to two sources/instruments/databases, one containing the data transmitted by satellite "A" and the other, by satellite "B". Usually, each database contains different products with several levels of processing, but all of them share the format with some common attributes and metadata.

As stated in [12], to have a meaningful collocation, the pixels from both images must have a physical overlap, which means that they need to meet a spatial and a temporal criterion within some threshold of error.

First, two files that may meet the time overlap criterion are selected and downloaded. For example: An Extractor retrieves an Image A that was stored in format A from Data source A, so the file can be named "ImageA.extA", where "extA" means the extension for file format A. The more common scientific file formats for Earth Observation data are NetCDF (`.nc`)[15], a version of HDF (`.hdf`), `.h5`)[9], and GeoTIFF (`.tiff`)[7].

Each Image is for our formalism a *Concept* characterised by the *Attributes*:

Time: The time at which each pixel of the image was acquired. It is usually provided in UTC or in some format of absolute time with a defined origin.

Geoloc: The geolocation of each pixel of the image, the point on Earth measured by the sensor. Each pixel is characterised by its spatial coordinates in some projection related to the type of orbit of the satellite.

Parameters: Every pixel of the image is characterized by n parameters. A parameter is a measurement taken by the satellite instrument or some quantity derived from it.

A *Transformer c* transforms the files into a common format so it will be easier to work with them and perform the collocations later. Formally

$$c : extA, extB \to extC$$

Then, the *Transformer f* gets the location of every pixel from image A as input and converts its coordinates from projection A to Projection B.

$$f : coordA_{projA} \to coordA_{projB}$$

Next, pixels that meet the spatial overlap criterion within some threshold of error are selected and can be collocated. *Transformer t* takes care of that task, taking the coordinates of geolocation, both in some projection, as input and retrieving a new product as output.

$$t : ParametersA, ParametersB \to PixelA\&B$$

The output is a *Concept* called "Pixels A&B", which format is the common format, and contains the Parameters from A and the Parameters from B, that were attributes of the pixels A and B and have not been altered or transformed. The entire process can be represented as

$$(c \circ f \circ t) : ImgA, ImgB \to PixelA\&B$$

Finally, a *Loader* loads the final product into a Database.

4 Results: An implementation

We have applied the discussed design in a Python Package that, a priori, aims to be used to collocate [12] data from a radiometer aboard a geostationary satellite with data from a radar aboard a polar-orbiting satellite.

For the source “A” of data we chose the ABI (Advanced Baseline Imager) on board geostationary GOES-16, with a central longitude of -76°, which allows it to take images of the whole American continent, with a temporal frequency ranging from 5 to 15 minutes [2]. The ABI is a multispectral radiometer that sense the Earth in 16 bands ranging from the visible to the NIR part of the electromagnetic spectrum [18]. An image of the whole continent is square and has a spatial resolution of 0.5 km to 2 km and a side size of 5424 to 16272 pixels, depending on the band. See table 4 for details about this. Every pixel of an image is geolocated and each file name contains information about the acquisition time of the measurements. The ABI data is stored in *NetCDF* format hosted in Amazon Web Server (AWS) [15].

Bands	Resolution (Km)	Image size (pixels)
2	0.5	16272
1, 3, 5	1	10848
4, 6-16	2	5424

As source “B” we choose the CPR (Cloud Profiler Radar) on board the polar satellite CloudSat. Its orbit is polar sun-synchronous, with a temporal frequency of 16 days. The CPR was designed to generate vertical profiles of clouds every 1.1 km along-track and the spacial resolution of each data point is 1.3 km across-track and 1.7 km along-track [20]. Every data point is associated with the geolocation and time of the measurement. One of the most interesting things about this mission is a product which provides the type of cloud among and other variables for each vertical profile [17]. The data acquired by CPR is stored in the CloudSat Data Processing Center, which is a SFTP server [1], in HDF-EOS format[9].

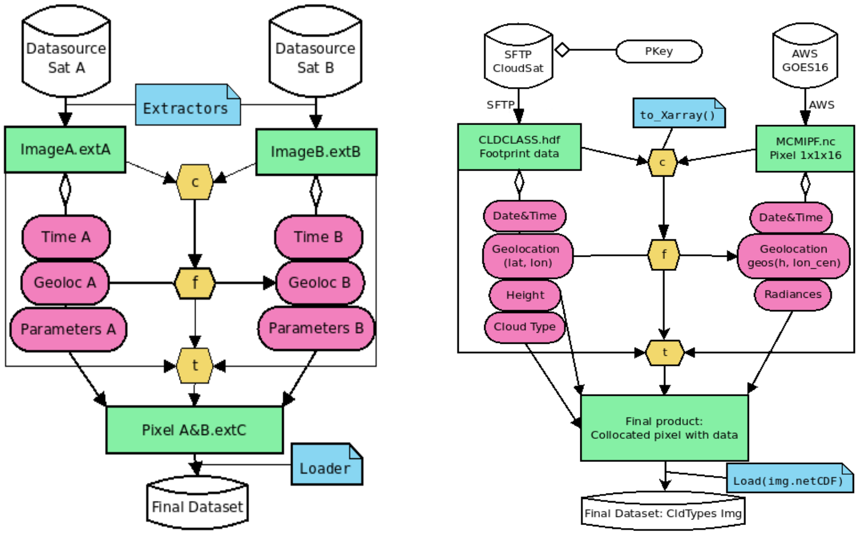
With the aforementioned descriptions, and based on the theoretical model and nomenclatures presented in the previous section, we have constructed a flowchart (Figure 2) that specifies what the code does.

Figure 2b tells the same story as Figure 2a but in different terms, regarding the implementation. The idea of the pipeline consists in extracting by means of a SFTP service from database A an image of an orbital passage of the CloudSat satellite (*concept*), which contains in each pixel the *attributes* date, time, geolocation (latitude, longitude), height in meters and type of cloud for each height.

Once the user has selected a time range in the CloudSat extracted data, this range information is fed to the AWS extractor which retrieves a suitable multiband image from GOES16. This image is also a *Concept* and in this case, each pixel has the *attributes* date, time, geolocation in geostationary projection (central longitude and height of the satellite) and radiance.

To know which CPR pixels or data points correspond with which ABI pixels, the Transformer f makes the coordinate change:

$$f : (lat, lon) \rightarrow geos(h, lon_c, R_e, R_p)$$



(a) Conceptual design. (b) Stratopy workflow.

Fig. 2: Structure diagrams of the proposed pipeline built with Stratopy. On the left side, there is a diagram containing the conceptual parts, and on the right side an implementation with the components provided by the project.

where *geos* is the projection used by GOES16 to georeference each pixel (see [21], section 5.1.2.2 for more information). This transformation depends on GOES16 height h, central longitude lon_c, equatorial radius R_e and polar radius R_p. This information is used to collocate pixels from the multi-band image with pixels from the CloudSat pass and then retrieve a product where every collocated pixel contains information on the radiance, the height of the atmosphere and the cloud types found in them.

Finally, the module Loader would take care of loading this final product into storage. This module is not defined yet, but we plan to implement it as an extension of some workflow orchestration program such as Apache Airflow [11] or Dagster [14]; All these technologies are based on the creation of tasks on an acyclic-directed graph (DAG) that allows fragments of the pipeline to be executed automatically in parallel or sequentially as required.

For your convenience, a prototype of this pipeline can be found in the Stratopy package [16]. Please note that the project is under active development and the current state can be explored in the project repository https://github.com/paula-rj/StratoPy/tree/dev.

5 Conclusion and further work

The ETL formalism was very useful to achieve an orderly design of workflows.

The modular structure of the concepts and attributes allows extending extractors and transformers in a simple way to extract and transform data from any of the currently active and most used Earth Observation satellites, such as GOES16/17/18 [10], Himawari [3], etc.

Also, the existence of some large ecosystem of remote sensing data analysis and data analysis packages in Python in general, and SatPy[5] for manipulation and transformation of data from remote-sensing earth-observing satellite instruments in particular, and Stratopy is a missing piece for orchestrating these transformations and analysis.

The future work, part of it is already started and consists of the orchestration of the processes already implemented in some kind of DAG framework such as Apache Airflow or Dagster.

References

1. Cloudsat data processing center. www.cloudsat.cira.colostate.edu, accesed: 04/04/2023
2. GOES-R. https://www.goes-r.gov/, accessed: 07/03/2022
3. New geostationary meteorological satellites — himawari-8/9 —. https://www.jma.go.jp/jma/jma-eng/satellite/news/himawari89/himawari89_leaflet.pdf
4. Booch, G., Rumbaugh, J., Jacobson, I.: Unified modeling language user guide, the (2nd edition) (addison-wesley object technology series). J. Database Manag. **10** (01 1999)
5. Boudriki Semlali, B.E., El Amrani, C.: Big data and remote sensing: A new software of ingestion. International Journal of Electrical and Computer Engineering **11**, 1521–1530 (04 2021). https://doi.org/10.11591/ijece.v11i2.pp1521-1530
6. Caserta Joe, K.R.: The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data. (1965)
7. Devys, E., Habermann, T., Heazel, C., Lott, R., Rouault, E.: OGC GeoTIFF standard. Tech. rep., Open Geospatial Consortium (2019)
8. Eliasson, S., Raspaud, M., Dybbroe, A.: Distributed Earth-Observation satellite data processing with Pytroll/Satpy. In: EGU General Assembly Conference Abstracts. p. 13133. EGU General Assembly Conference Abstracts (May 2020). https://doi.org/10.5194/egusphere-egu2020-13133
9. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the hdf5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. pp. 36–47 (2011)
10. Galica, G., Dichter, B., Tsui, S., Golightly, M., Lopate, C., Connell, J.: Goes-r space environment in-situ suite: Instruments overview, calibration results, and data processing algorithms, and expected on-orbit performance. In: Earth Observing Missions and Sensors: Development, Implementation, and Characterization IV. vol. 9881, p. 988118. International Society for Optics and Photonics (2016)
11. Harenslak, B.P., de Ruiter, J.: Data Pipelines with Apache Airflow. Simon and Schuster (2021)

12. Holl, G., Buehler, S., B, R., Jimenez, C.: Collocating satellite-based radar and radiometer measurements - methodology and usage examples. Atmospheric Measurement Techniques **3** (06 2010). https://doi.org/10.5194/amt-3-693-2010
13. Nagpal, A., Gabrani, G.: Python for data analytics, scientific and technical applications. In: 2019 Amity International Conference on Artificial Intelligence (AICAI). pp. 140–145 (2019). https://doi.org/10.1109/AICAI.2019.8701341
14. Ooi, W.T.: Dagster: Contributor-aware end-host multicast for media streaming in heterogeneous environment. In: Multimedia Computing and Networking 2005. vol. 5680, pp. 77–90. SPIE (2005)
15. Rew, R., Hartnett, E., Caron, J., et al.: Netcdf-4: Software implementing an enhanced data model for the geosciences. In: 22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology. vol. 6 (2006)
16. Romero, P., Robledo, J., Villa, J., Rosales, G.: Stratopy. https://github.com/paula-rj/StratoPy.git, accesado: 26/06/2020
17. Sassen, K., Wang, Z.: Classifying clouds around the globe with the cloudsat radar: 1-year of results. Geophysical Research Letters **35**(4) (2008). https://doi.org/https://doi.org/10.1029/2007GL032591, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2007GL032591
18. Schmit, T., M.M., G., W.P., M., J.J., G., J., L., A.S., B.: Introducing the next-generation advanced baseline imager on goes-r,. Bulletin of the American Meteorological Society **86**, 1079–1096 (2005)
19. Semlali, B.E.B., Amrani, C.E., Ortiz, G.: Sat-etl-integrator: an extract-transform-load software for satellite big data ingestion. Journal of Applied Remote Sensing **14** (2020). https://doi.org/10.1117/1.JRS.14.018501
20. Stephens, G.L., Vane, D.G., Boain, R.J., Mace, G.G., Sassen, K., Wang, Z., Illingworth, A.J., O'connor, E.J., Rossow, W.B., Durden, S.L., Miller, S.D., Austin, R.T., Benedetti, A., Mitrescu, C.: The cloudsat mission and the a-train: A new dimension of space-based observations of clouds and precipitation. Bulletin of the American Meteorological Society **83**(12), 1771 – 1790 (2002). https://doi.org/https://doi.org/10.1175/BAMS-83-12-1771, https://journals.ametsoc.org/view/journals/bams/83/12/bams-83-12-1771.xml
21. Valenti, J.: GOES-R Series 416-R-PUG-L1B. 0347 Vol 3 Rev 2.1. . Tech. rep., NOAA (11 2019)
22. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual modeling for etl processes. In: Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP. p. 14–21. DOLAP '02, Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/583890.583893, https://doi.org/10.1145/583890.583893
23. Zantedeschi, V., Falasca, F., Douglas, A., Strange, R., Kusner, M., Watson-Parris, D.: Cumulo: A dataset for learning cloud classes (11 2019)