Software Design Document

February 17th, 2023

Version 1.2

USGS AirFlow Processing Pipeline

Team ARES

Sponsored By:

Trent Hare

Team Mentor:

Vahid Nikoonejad Fard

Team Members:

Hunter Woodruff (Team Lead)

Quinton Jasper

Chris McCorkle

Isaiah Raspet

Richard McCormick

Table of Contents

I. Introduction

With each success that NASA accomplishes, landing rovers onto the surfaces of foreign planets and conducting groundbreaking research, it's easy for the average person to overlook the crucial underlying sciences that support such efforts, especially in the early stages. Questions such as: “How can this rover land safely on planet Y?” or “How will this rover navigate the surface of planet Y once it is landed?” require reliable answers before any technology can set foot in outer space. NASA is not alone in its effort to answer such questions.

The United States Geological Survey (USGS) Astrogeology Science Center is a government research center based in Flagstaff, Arizona. Their mission is to collect, analyze and publish cartographic and geological data, which helps both researchers and the general public to better understand the surface of not just Earth, but also other celestial bodies within our solar system. The USGS team accomplishes this via multiple pieces of software designed for specific tasks, such as file conversions, data compression, image rendering, etc. For the course of this project, Team Ares will be working with a specific package of USGS software, collectively known as ISIS3.

ISIS3 is a package of more than three-hundred individual applications that each handle unique roles in the image processing workflow. The sponsor for this capstone project, Mr. Trent Hare, has informed the team that, while ISIS3 is a complete product and works for their needs, its fragmentation makes it difficult for users to manage. To follow the workflow properly, users have to locate and understand each required application from within a Linux terminal. It is also expected that users will need to establish custom workflows, where each may consist of many different combinations of applications based on the user’s needs. From the perspective of a researcher or an enthusiast, the current structure involves too many steps with too many complications that increase the time and energy required to obtain the desired outputs.

Our team has been tasked with assisting in the development of *a workflow management system*. That is a system that makes the process of organizing, executing, and monitoring a workflow more intuitive and accessible to more users. To our sponsor, this would be best accomplished with some sort of graphical user interface. This interface would allow users to interact with their workflows much like a flowchart. This *Directed Acyclic Graph* (DAG) would consist of nodes

that represent an individual application, and the connections between these nodes show the direction of data input and output.

This document will outline both our team's vision for the final product, as well as the implementation of the various components required to build it. These include summaries of the proposed implementation, proposed architecture, as well as the modules and interfaces which are desired by the Client to be present in the end product. We will then conclude with a detailed plan of how each element is to be implemented by our team.

II. Implementation Overview

The overall vision of this project is to provide the client with a clean, user-friendly, and visually informative software that will allow users to design Directed Acyclic Graphs (DAGs) with graphics-based tools, easily deploy and run those graphs in a pipeline, and to be able to visually monitor those pipelines until completion. To accomplish this, we will make use of several software tools, including Apache AirFlow, Elyra, ISIS3, Kalysiris, and Anaconda. The end result will be a single software package that can be installed on a user's workstation, or into a cloud environment, allowing users to construct, run, and monitor workflow pipelines with minimal training.

At the core of this project rests ISIS3, the Integrated Software for Imagers and Spectrometers version 3. This software allows different filtering and processing modules to be applied to raw data collected from remote sensing arrays. Initially, our project will rely on converting each ISIS3 module into a format that is compatible with Apache AirFlow. This conversion process will rely on the Kalysiris package, which gives a Python-based interface for each module in ISIS3. Our minimum viable product will consist of combining the ISIS3, Kalysiris, and AirFlow softwares together in such a way that allows users to construct pipelines from ISIS3 modules and run them in the AirFlow environment.

With the minimal viable product completed, focus will shift towards completing the stretch goals, which will round out and finish the project. Among these stretch goals is the integration of Elyra, a software that allows DAG's to be visually constructed with a drag-and-drop interface. Modules from ISIS3 will be directly imported and can be used within Elyra to build pipelines that can be directly exported to and run within AirFlow. The end goal of this integration is to contain Elyra within the original software package containing AirFlow, ISIS, and Kalysiris, such that Elyra can also be deployed with the minimum viable product.

The last stretch goal of this project is to integrate our software package with the online web map for ISIS, which allows users to view their finished product directly on the planetary body from which the data was originally captured. As the most technically challenging portion of the project, this goal has been designated as the final and least critical milestone, as it will not have any effect on either the minimum viable product or the other stretch goals. In essence, the

final product generated by the user will be directly exported from Apache AirFlow into an environment where it can be displayed using the CartoCosmos plugin for Leaflet, a web-based map viewer.

The final vision for this project, taken from the requirements supplied by the client, will be a simple-to-use, aesthetically pleasing tool which allows users to construct and run custom pipelines from the ISIS3 software and display their finished product onto a map for public distribution.

III. Architectural Overview

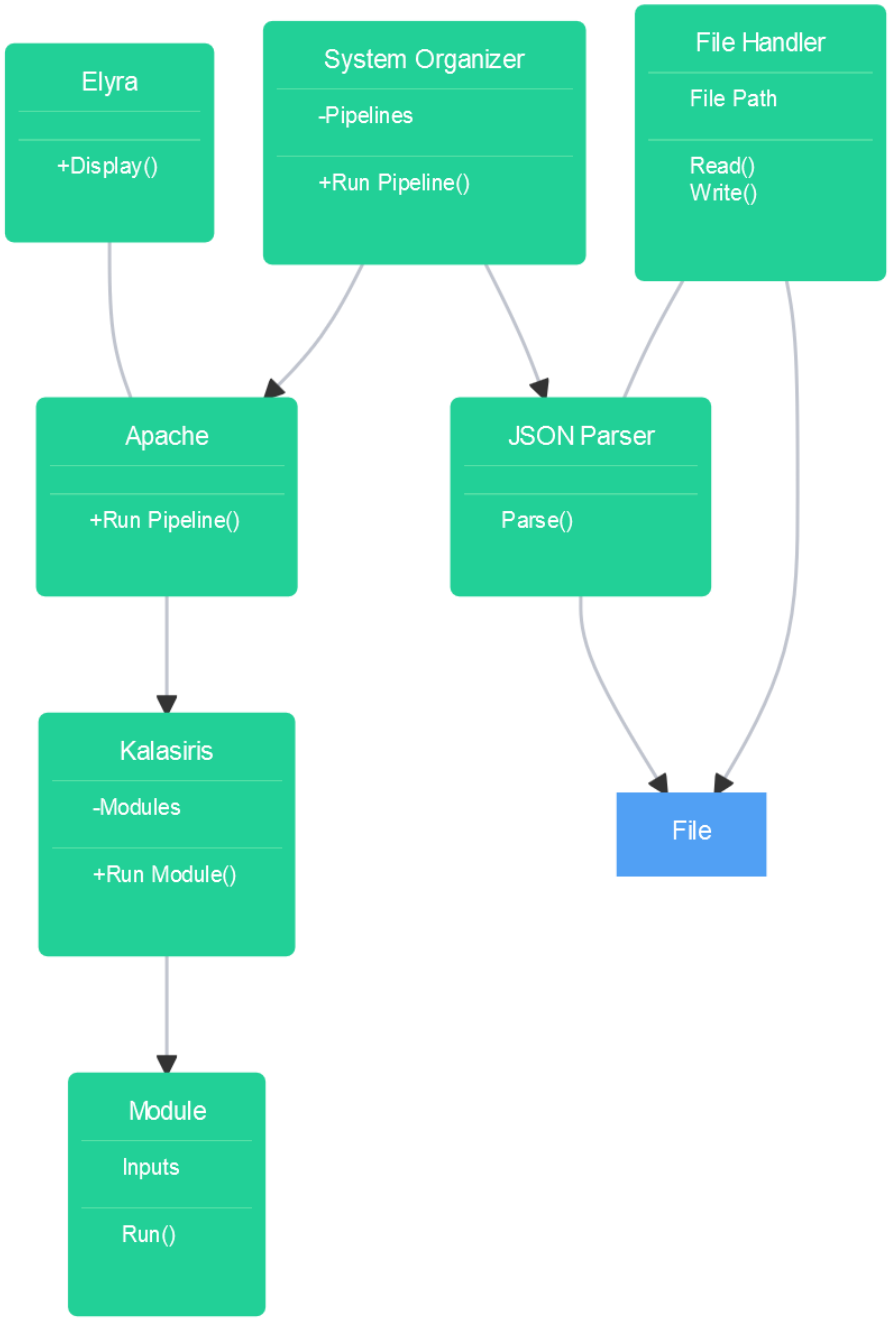
The architectural focus of this project is on simplicity. This is shown by the UML diagram overview of the project. Each module handles a specific task and hands whatever data is needed to a different dedicated module. The main module of this project is the system organizer, which will deal with directing requests to the specific modules that need them. For example, if, through Apache, a user requests to load and run a specific pipeline. The system organizer will take this request to the file handler, which will load the relevant JSON file which will then be parsed by the JSON parser.

As shown in fig. 1, the overall architecture of the project is very simple. There are a few stubs including “module” and “file” which represent different repeating aspects of the project. The “module” will in reality be a series of modules specific to ISIS with their own inputs in the form of filenames and outputs, also in the form of file names. The only files handled by the File Handler are those that are JSON files that represent pipelines. This is how pipelines will be saved and loaded from the user’s hard drive, as well as run through Apache. The rest of the file handling is done internally inside ISIS.

What is not easily depicted here is the actual amount that will solely be done in Apache. Aside from loading JSON files into Apache pipelines, not much interaction will need to take place on that side of the system. Along with Elyra, as long as we are able to create nodes with defined functionality and input, much of the pipeline creation will be done within Apache, as that should be much easier for the user to be able to accomplish. The inclusion of the ability to parse and write JSON is a legacy addition, which should make this software entirely compatible with the process through which things used to be done.

It should be noted that, because of the current design philosophy, there is not much static state contained within this system. This is due to the fact that the entire system is involved in running existing software and is able to reify and save the state that said software was run in. Because of this saving of a pipeline to the disk, the state does not need to be remembered within the system statically, only referenced. Doing so will keep program loading times low, as pipelines are only loaded and executed at the request of the user, but also help to keep the code as uncluttered as possible. As stated above, the goals of the architecture of this project are to

keep complexity down for both the users and programmers.



(Fig 1)

IV. Module and Interface Descriptions

System Organizer

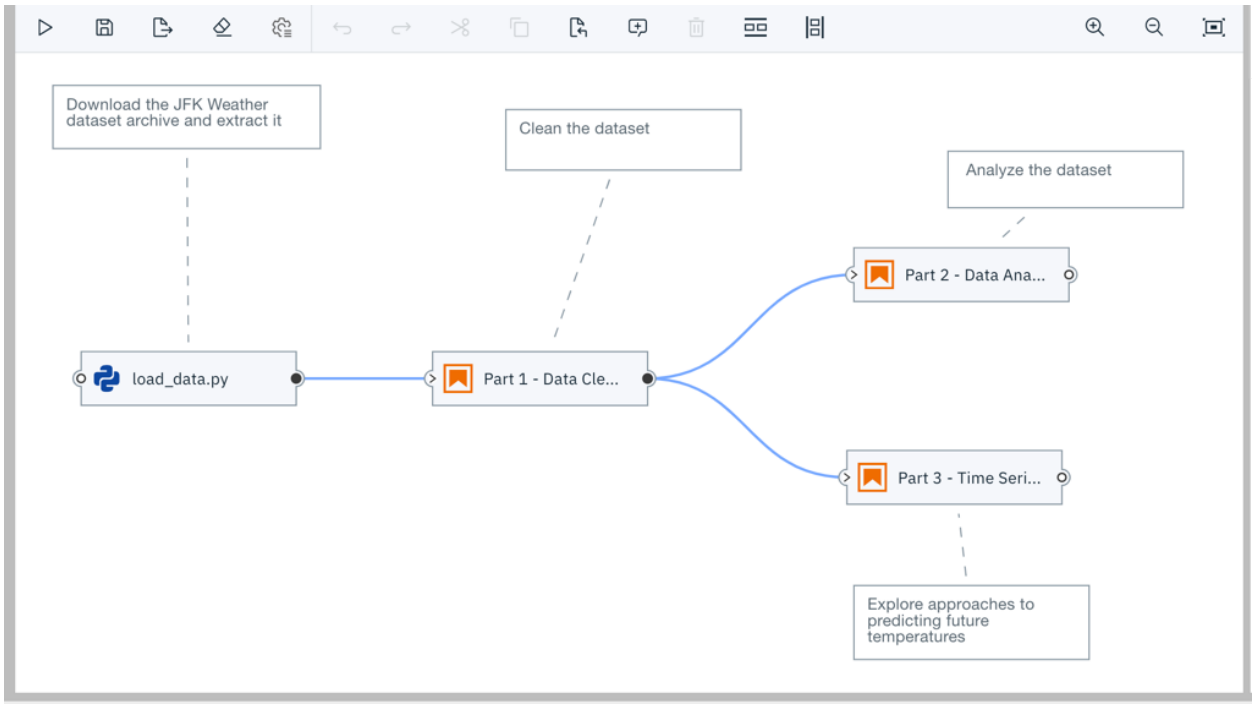
The system organizer's function is to ensure that all modules are functioning together cohesively. Its goal is to direct requests to the proper module so that everything runs smoothly. This system organizer is required for the project to follow principles of encapsulation and object-oriented programming. It stops adjacent modules from needing crossed functionality, and instead of repeating code, a module that handles a single task will be called to run whatever functionality is necessary. The system organizer has direct oversight of Apache and the JSON parser, each of which is the only components necessary to direct requests from the user.

Elyra and Apache AirFlow

The pipelining software Apache AirFlow is a free software that will act as the heart of our system organizer. AirFlow will allow the user to create, save, and easily use pipelines which will be imported into AirFlow via a JSON file. With this, researchers can create DAGs (Directed Acyclic Graph(s)) to efficiently run ISIS3 commands wrapped in python, via the use of a pipeline. Functionally, this means that AirFlow adopts a modular and replaceable interface wherein researchers can edit and save pipelines to complete specific tasks, without having to run base Linux commands every time to receive an actionable product. Along with the base functionality of the AirFlow pipelining software, Elyra will be introduced as well to supplement some of the ease-of-use restraints in AirFlow.

Elyra, at its core, is a simple add-on or modification of the base AirFlow software. What Elyra adds is the ability to "drag and drop" our python-wrapped ISIS3 commands – henceforth referred to as nodes. With Elyra, AirFlow can be used in the simplest way possible for researchers that may not be familiar with more "low-level" interface functionality, like running commands. Elyra provides a **Pipeline Visual Editor** for building pipelines from notebooks,

Python scripts, and R scripts, simplifying the conversion of multiple notebooks or script files into batch jobs or workflows. For our uses, the interface Elyra provides will allow researchers to build pipelines using the aforementioned UI simplifications (“drag and drop”, etc) to create a consistent, reproducible, and simple workflow. An example of an Elyra / AirFlow workspace and pipeline is pictured below (Fig. 2)



(Fig. 2)

Overall, the system organizer will be able to integrate these technologies easily so they can interact in an abstracted pipeline to create actionable products at the click of a button. The JSON Parser will feed input data into AirFlow which will employ Elyra, and the pipelines will directly interact with ISIS3. The result of this interface will be a USGS useable product, allowing the team to efficiently deploy these technologies to their needs.

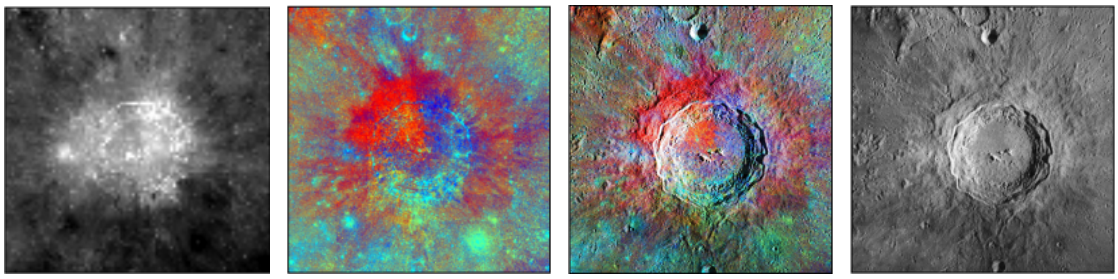
JSON Parser and File Handler

The JSON parsing module will be implemented as a core part of the system organizer. This module will allow the application to open XML-defined “recipes” which contain the specifications for a user-defined data pipeline. This allows relevant information to be imported into Apache, allowing it to implement the correct ISIS modules in the correct order to process the given image within user specifications.

The file handler will coordinate this activity. Responsible for managing not only the “recipe” files, but also the input and output files, this module will allow the user to store their inputs, retrieve their outputs, and to also access each recipe from a stored file that they might need. This module will thus provide a level of abstraction that will allow the user to most efficiently use the application, while preventing them from having to deal with unsightly or unintuitive terminal operations. These modules will not have an interface, but will operate from behind the scenes to ensure the best quality experience for the user.

ISIS and Kalasiris

Integrated Software for Imagers and Spectrometers or ISIS is free and open-source software developed by the USGS Astrogeology Science Center for NASA and the planetary hobbyist community. ISIS is a tool that leverages standard image processing techniques such as contrast, stretch, image algebra, and statistical analysis for processing raw data into analysis-ready products. Unlike other image processing tools, ISIS’ key feature is its ability to compile different types of data to create archives, topographic or cartographic maps, digital elevation models, and other scientific products that can be used in any number of applications. ISIS3 is the core of the AirFlow Processing Pipeline, with 300+ modules that handle all of the image processing. It is the sole image processing software needed. ISIS has a single dependency, Kalasiris, which is another major aspect of the AirFlow Processing Pipeline.

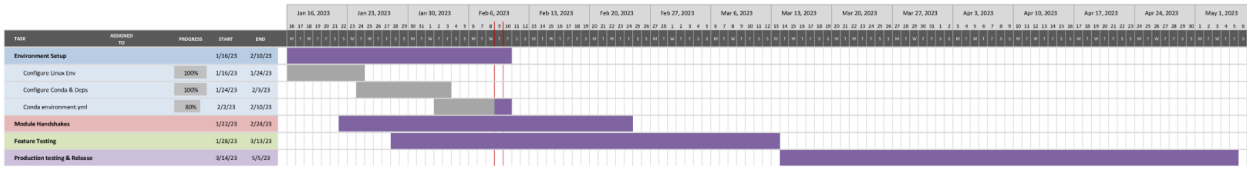


(Fig. 3) Example of image processing process done by ISIS

Kalasiris is what is known as a wrapper. A wrapper is a supplementary software that wraps, or compiles, the functions and functionalities of another software so they can be leveraged more efficiently. In the case of Kalasiris, it was specifically built as a lightweight wrapper for the ISIS software. Its lightweight build allows for the wrapping of the python subprocess modules necessary for the easy calling of ISIS with no python dependencies. Kalasiris does however require an instance of ISIS to be installed on the system in which Kalasiris is trying to be used. Without Kalasiris, ISIS would be a loose collection of image-processing tools unable to be integrated with projects such as the AirFlow Processing Pipeline. As such Kalasiris is the bridge between ISIS and the greater AirFlow Processing Pipeline ecosystem listed above.

In past and current implementations of ISIS and Kalasiris, users would have to interface directly with these softwares. It is important to note that at this time there is no dedicated user interface so users are required to use the console of their system to call the wrapped scripts necessary to process the images. Since the beginning of the AirFlow Processing Pipeline project, USGS has hoped that the necessity to interface directly with these softwares be done away with in place of a user-friendly interface that is faster to learn and easier to use. As such these components will not have a public-facing interface and will instead be leveraged by Apache AirFlow and Elyra to provide a better user experience.

V. Implementation Plan



Culminating with the work that was done in the previous term, our team has designed a modular solution for our client. As such, we find it only appropriate to design our implementation plan around the completion of these modules, which will be categorized within separate phases. Each phase represents major development milestones consisting of smaller, low-level tasks. The development phases are as follows: phase one - environment setup, phase two - module handshake development, phase three - feature testing, and phase four - production testing & release. These phases will be expanded upon below.

Environment setup is a fundamental development phase that will allow the team to establish a common and reproducible working environment. This phase contains the required tasks of: Choosing a standard Linux environment for development, configuring Anaconda to manage project dependencies, and making the Anaconda environment reproducible via an *environment.yml* file. Many of these conversations have already been had in the previous term, though some decisions still require finalization, requiring the inclusion of this phase within the current term. As a team, we have allotted a brief period of roughly 3 weeks at the beginning of this term to allow the finalization of the chosen Linux Mint development environment.

The ‘Module Handshakes’ phase contains tasks relating to the configuration of each individual module. Each module serves a purpose in the greater system, and thus appropriately communicates with one another to get work done. As detailed above, these modules consist of Elyra, Apache AirFlow, and Kalasiris/ISIS3. This also includes helper libraries for JSON parsing and file I/O management on the local system. This milestone is among the more heavy, with the team allotting upwards of 34 days to complete. By the tail end of this milestone, we expect the product will, at a minimum, contain all logic required to demonstrate the requirements to satisfy the definition of our client’s Minimum Viable Product. Included in these requirements is the

ability for users to use a Graphical User Interface to construct DAGs, as well as to run and monitor pipelines using Apache AirFlow.

Testing and release phases mark the last major milestones in our development process. While the logic exists and the software is present, the team needs to take the time to ensure this software product best conducts the tasks as the client intends. Much of this phase will mostly be done in tandem with the previous phase. The most notable tasks that need to be completed within this phase is the development of sample input and output data, following guidelines and sample data provided by our client. Abstracting away the main product, Apache AirFlow and its system dependencies, our client requested our team to create two sample “recipes” in conjunction with our software product. By producing these recipes, we allow ourselves to automate tests for input and output data, as well as provide a template for our clients, such that they can develop their own recipes when the project has been turned over to them.

VI. Conclusion

The hard work provided by the team at the USGS Astrogeology facility has far-reaching applications that will affect the future of human space travel and exploration. However, before large-scale goals can be accomplished, optimizations must be made at a lower level in the system to increase efficiency and effectiveness. In this case, the USGS's original workflow system for processing and publishing gathered planetary data has consisted of a complex, manual execution process for all steps involved. Team ARES is tasked with developing a *pipeline management system* to allow researchers at the USGS to streamline their process of image processing and data publishing.

After dedicating time in the previous term to designing our solution, the team's lead architect has taken the incentive to carefully research and compartmentalize the development process into easily-workable modules. Above, we discussed how Elyra and Apache AirFlow will act as the project core, having already implemented pipeline visualizations and pipeline management; The ISIS3/Kalisiris packages will provide an API to connect our client's existing product to AirFlow; lastly, lower-level system components including a JSON parser and file I/O handling helpers will be included to enable the use of user-created pipeline "recipes" to be provided to the software.

Above, we have also established a loose plan of action to help the team stay on track to reaching high-level targets. Four main milestones were established, those being: Environment Setup, Module Handshakes, Feature Testing, and Final Production Testing & Release. These milestones will grow to be further elaborated as new challenges arise and solutions are developed.

Based on the progress our team has witnessed over the last couple months, we anticipate that the core development of our solution will be completed with sufficient time for crucial post-development checks, like user testing and improving our user's overall experience. Our development team is expected to be able to document the usage, and likely extend the functionality of the product via implementation of our client's stretch-goals.