User **S**ystem of **A**strogeology **T**echnologies

(**USAT**)

Design Specification

Kyle Andrew McGinn, Megan Backus, Zack Ellett, Mikal Ustad

Rev 3.4

3/5/2012

Table of Contents

Website:
http://www.cefns.nau.edu/Research/D4P/EGR486/CS/12-Projects/USAT/

Preface: Defined words are superscripted [a-z] at every occurrence and the subscripts can be found alphabetically at glossary near the end of this document for reference.

Introduction

ISIS[h] (Integrated Software for Imagers and Spectrometers) is a massive project of the USGS[p] (United States Geological Survey) that has been evolving since the 1970s. Its objective is to process images taken of space and celestial bodies into data to be used for analysis. It has grown into a software package that includes almost 300 separate programs. Each of these programs is designed to manipulate a given image in one certain way (e.g. one program removes spectral lines, another rotates the image). ISIS applications are currently used by calling a command on the command line, although some of them have a simple graphical interface. Even though ISIS has been used for over 40 years, it still lacks a single, cohesive interface to link all of its entities together.

The purpose of the USAT[o] (User Systems of Astrogeology Technologies) project is to remedy this; to create one centralized[a] GUI[g] (graphical user interface). All 300 ISIS[h] programs will be accessible through this one program, along with any new ISIS[h] applications created in the future. A user will also be able to string multiple programs together in a work flow with intuitive ease. This will allow the implementation of different ISIS[h] applications to one image. Currently if a user wishes to do this they must initiate each program manually or write their own script do to so.

Easing the use of ISIS[h] is the main goal for creating the USAT[o] GUI[g], a byproduct of which has produced a secondary goal. It currently takes about three to five months of training for a new employee to become productive in the use of ISIS[h]. While the creation of an intuitive GUI[g] can reduce that instructional period, the USAT[o] team intends to expand the available resources within the GUI[g] to reduce it even further. The GUI[g] will contain a Help Center, listing all ISIS[h] applications and a description of each. The description will include a synopsis of what the program does and provide an example if applicable.

The USAT[o] team has decided to utilize Galaxy[f]. Galaxy[f] is an open source, web-based program that aids a user in developing workflows. The current Galaxy[f] framework[e] will need to be modified to work with the ISIS[h] programs. Other challenges for the project include preparing for future expansions. The actual processing of ISIS[h] requires a lot of computing power, but interaction with it should not. Impending evolutions of ISIS[h] might enable a user to initiate ISIS[h] through a laptop while out in the field or even from a smart phone. However, one of the biggest challenges is the design of this GUI[g]. It must be intuitive, efficient, and generated with creativity and new ideas. GUI[g] design is the main focus of the USAT[o] project, and its elaboration the main purpose for this document.

Architecture Overview

The purpose of ISIS[h] is to manipulate imagery collected by current and past NASA[j] planetary missions, such as those sent to Mars, Jupiter, Saturn, and other solar system bodies. The system of programs that make up ISIS[h] are combined to create workflows which assist scientists in analyzing and manipulating this imagery. In an effort to improve the usability and fluidity of workflow creation, a framework[e] of tools called Galaxy[f] will be extended to integrate with ISIS[h].

Galaxy[f] was originally developed for bioinformatics research. However, the underlying framework[e] is easily extensible. Each tool is defined inside an XML[r] file where options and parameters are laid out in a way that is easily parsed by Galaxy[f]. This is very similar to how the tools in ISIS[h] are defined, so importing new tools into Galaxy[f] is a simple matter or "translating" from one form of XML[r] to another. An XML[r] tool builder will make this translation possible.

Previous design ideas oriented towards client architecture did not provide the reliability needed to operate in a changing environment. A centralized server gives us the reliability we need through dedicated servers with redundant data arrays and protection from a single point of failure by using uninterrupted power supplies. Likewise, the client in Client-Server provides the same reliability by being universal across all operating system platforms; which reduces error because no installation is needed. Switching from client to Client-Server also provided the ability for remote processing and cross-platform operation.

Galaxy[f] runs on a highly customizable client-server[b] architecture. The default instance runs on a standalone local web server, meaning the entire Galaxy[f] instance is completely portable. Furthermore, Galaxy[f] can be run as a normal web server, meaning that any machine with a web browser can access it. This eliminates many of the cross-platform shortcomings of the existing ISIS[h] interface. The customized website layout gives the user the ability to drag and drop tools onto a workflow. These tools are connected to other tools with varying numbers of inputs and outputs. Functioning workflows will be exported as executable scripts, preserving the original functionality of previously written ISIS[h] scripts. Additionally, Galaxy[f] has provisions for saving and sharing workflows in a much more user-friendly format than attaching a script to an email. All of this functionality is built-in to Galaxy[f] and will provide a solid foundation for future work in the area.

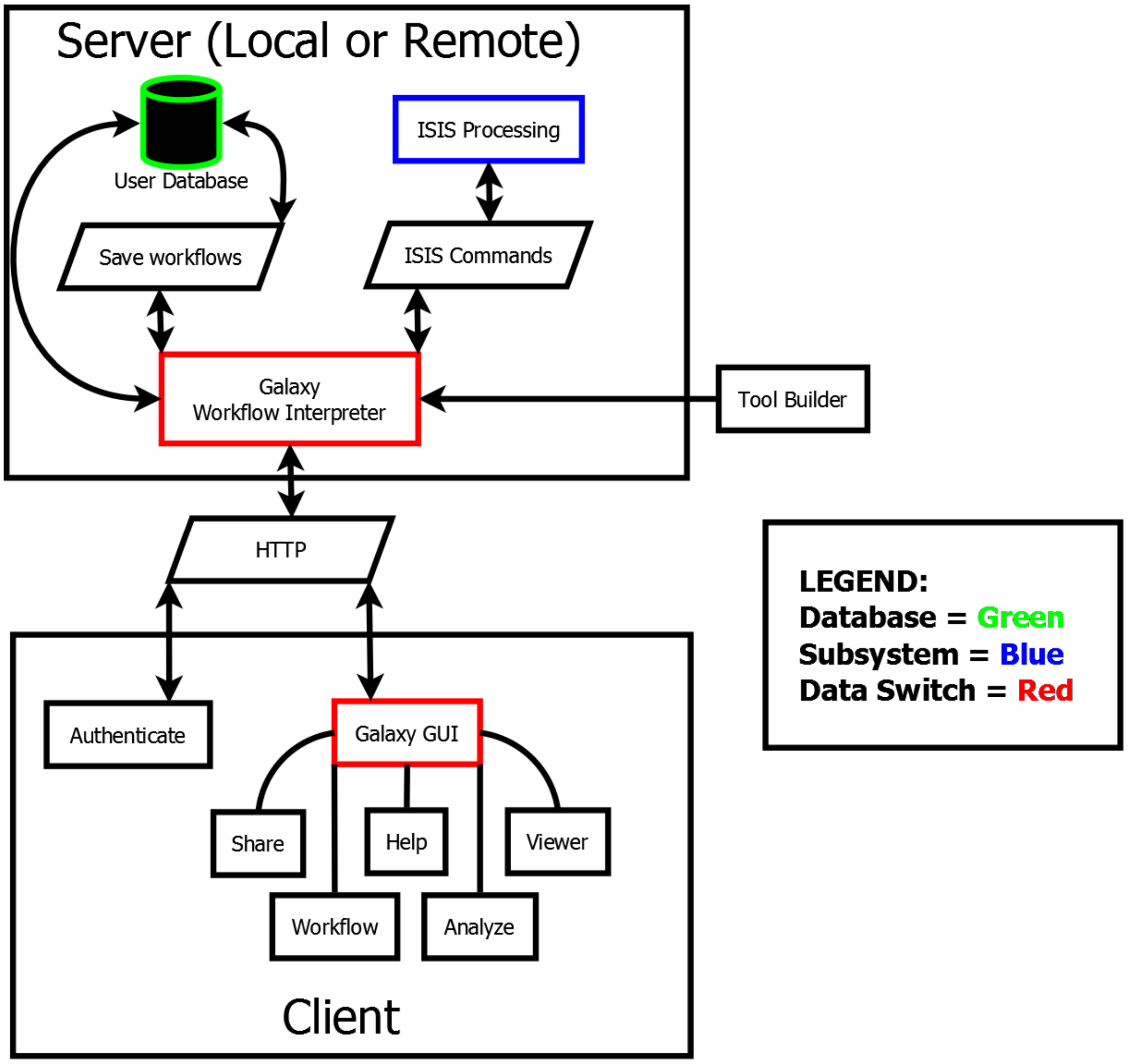


Figure 1: Client-Server Architecture Overview

As illustrated in Figure 1, the Client-Server[b] architecture communicates through an HTTP connection that is universal between all operating systems. For a remote server, the client simply connects via web browser anywhere with Internet access. Alternatively, the server can be installed locally on any Unix system and accessed via web browser. The difference between connecting to a local or remote server is where the User Database is stored; locally or remotely.

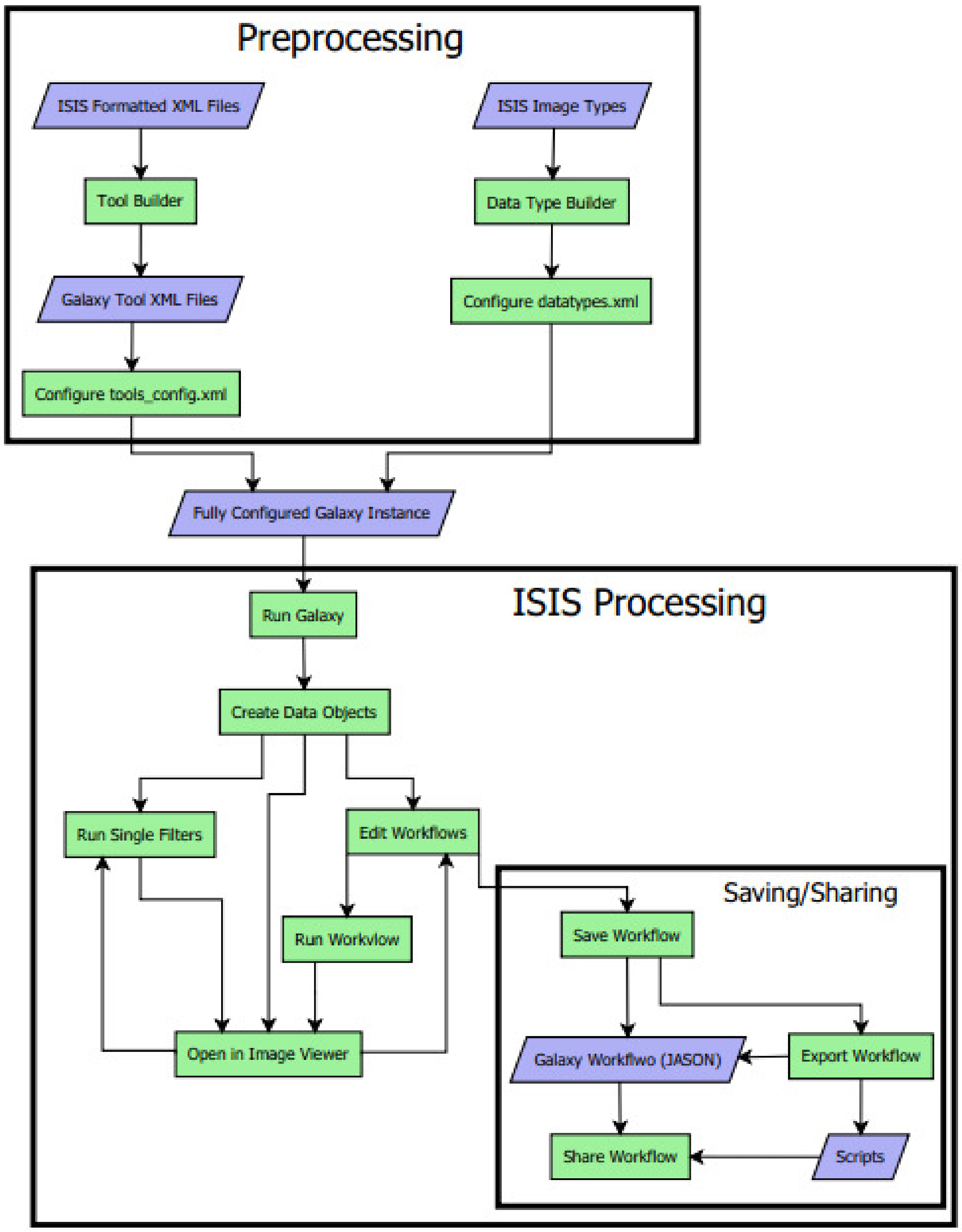
The Client is a GUI[g] that authenticates, sends commands to the server, and receives data from the server through the HTTP connection. The Client GUI[g] has the following tabs: Share, Workflow, Help, Analyze, and Viewer. The Server has 4 functions: receiving workflows from Client, storing workflows, sending workflows to be processed by ISIS[h], and sending database information to the Client.

The USAT[o] system is based upon a goal-oriented design. This encompasses maintainability[i], expandability[d], understandability[m], ease of use[b], reliability[k], and robustness[l]. Many of the anticipated changes to this system in the future will be automated and support the expandability[d] goal by only requiring the addition of new XML[r] files from ISIS[h] programs. Understandability[m] is achieved through the Help Module by having both simple and detailed descriptions. Having the option of a dedicated server gives the system reliability[k] over client architecture because it is less susceptible from a single point of failure. The client provides the same reliability[k] by being universal across all operating system platforms. The server increases extensibility through only needing a converted XML[r] file associated to each ISIS[h] program. Errors on the client side are also reduced because no installation is needed. Unknown or unexpected input into the USAT[o] GUI[g] will not cause the program to crash because the project contains safeguards against user error and incorrect parameters. Early user testing[p] and unit testing[n] will be utilized to find and correct errors.

This design benefits both the Galaxy[f] community and further ISIS[h] development. By using the methods described here to enhance Galaxy's[f] capabilities with regard to image processing, the Galaxy[f] community will no longer be constricted to bioinformatics. With the right tools implemented, Galaxy[f] can be setup to process any type of data with an easy to use GUI[g]. ISIS[h] users also benefit from this design because it offers a solid platform from which to implement further features into the system, including remote processing submission and mobile workflow creation.

Detailed Module Descriptions

Figure 2 shows a simplified use case for the entire system. It is used as a way to visualize all the functionality that is required of the USAT[o] system.



The USAT[o] system starts with ISIS[h] formatted XML[r] files and ISIS[h] image types. After the Tool Builder and the Data Type Builder are run the Galaxy[f] Server Instance will be configured and ready to serve clients.

The clients will then connect to the server and begin performing ISIS[h] image processing tasks. The first step in this process will be to Create Data Objects, or images as objects that can be used as inputs to filters in the Galaxy[f] GUI[g]. Users can do this by either finding images already stored on the server, uploading new images from their local system, or downloading images to the server from other locations.

Once users have some data to work with they must decide what they will do next. They can choose to Run Single Filters on those images. After the filters are run they will have a new data object that they can either apply more filters to or view in the Image Viewer. If they wish to apply multiple filters at once they can create a Workflow using the data objects as the initial parameters. When they have completed their workflow they can Run it, Save it or both.

If they choose to run the workflow it will yield new data objects that they can view in the Image Viewer2.15 or process further. If they choose to Save the Workflow they have the ability to save it to their user profile, in which case it will be Saved as a JSON file that they can open again later or Share with other users. They also have to option to save as a number of different types of Scripts that can be run independently from the Galaxy[f] GUI[g].

The UML diagram below (Figure 3) describes what role each module outlined in the Architecture overview will play in making the USAT[o] system function like Figure 2 describes.

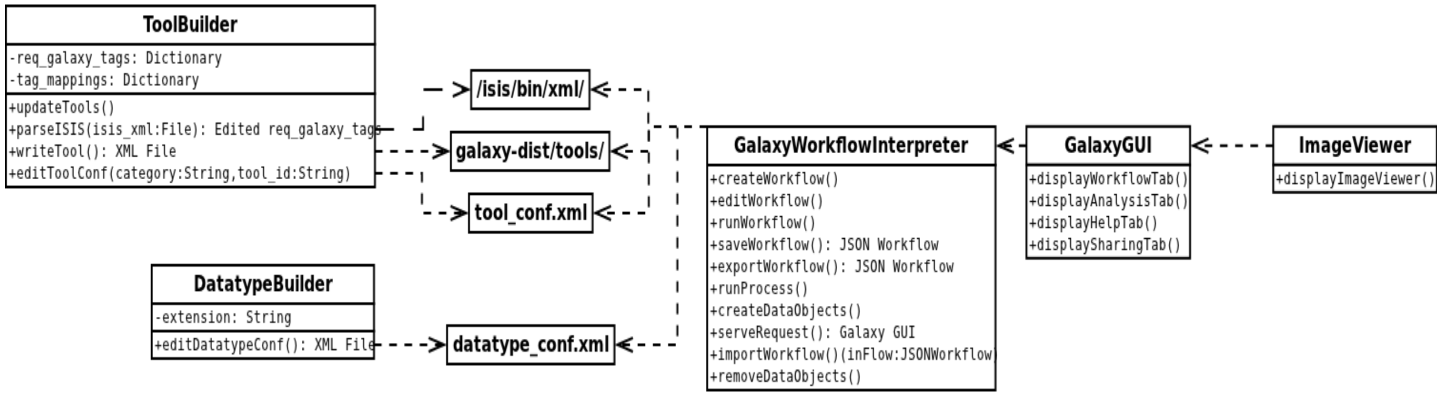


Figure 3: Class Diagram

ISIS[h] Software Suite

ISIS[h] Software suite consists of around 300 individual C++ programs that are run from the command line. No changes will be made to those programs and the USAT[o] system will interface with

ISIS[h] by calling the commands specified by the user in the GUI[g].

Tool Builder

The tool builder (Figure 4) will be located in the galaxy-dist folder and will be able to parse a directory full of XML[r] files in ISIS[h] format, convert them to Galaxy[f] formatted XML[r] files. Then it will drop the newly created XML[r] files in the correct galaxy-dist/tools/<category> folder based on the category parsed from the ISIS[h] file. Finally it will edit the tool_conf.xml so the new tool will be displayed and usable.

This will need to be run when a change is made to any of the ISIS[h] XML[r] files or when new tools are created.

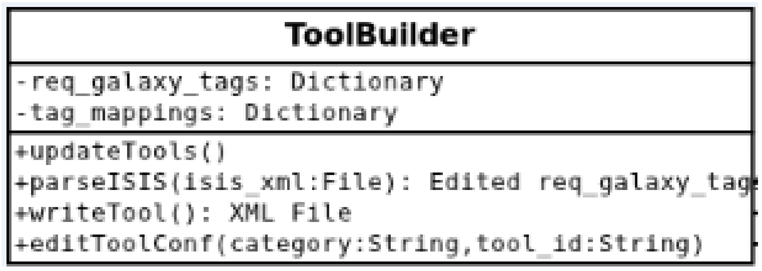

```
ToolBuilder
-req_galaxy_tags: Dictionary
-tag_mappings: Dictionary
+updateTools()
+parseISIS(isis_xml:File): Edited req_galaxy_tag
+writeTool(): XML File
+editToolConf(category:String,tool_id:String)
```

Figure 4: ToolBuilder UML

Fields

-req_galaxy_tags: Dictionary

Is a python[s] dictionary that will include all of the tags that are required for a valid Galaxy[f] tool XML[r] file, and map them to default values.

-current_tags: Dictionary

Is a python[s] dictionary that is the working copy of the required galaxy[f] tags.

-tag_mappings: Dictionary

Is a python[s] dictionary that will include all the possible tags found in an ISIS[h] formatted XML[r] file and map them to the Galaxy[f] tag they correspond to.

Functions

+updateTools():

Iterate over every ISIS[h] XML[r] file in /isis/bin/xml/ and call parseISIS(), writeTool(), and editToolConf() on each.

+parseISIS(isis_xml:File):

Takes in a single ISIS[h] formatted XML[r] file. First it resets the current_tags to the req_galaxy_tags, meaning all tags go back to default values. Then it parses the given file and uses the tag_mappings to edit the values of the current_tags dictionary. When it has finished parsing the file the current_tags dictionary should map the correct values from the ISIS[h] XML[r] file in the correct tags for a Galaxy[f] XML[r] file.

+writeTool(): XML[r] File

Uses the current_tags dictionary to write a Galaxy[f] formatted XML[r] file in the correct directory. The directory should be determined by the category, /galaxy-dist/tools/<category>/<new_XML_file>. Should overwrite file if it already exists in case changes were made to the ISIS[h] file.

+editToolConf():

Edits the tool_conf.xml file located in /galaxy-dist/ so that it includes the tool that was placed in the tool folder by writeTool(). It will put it under the correct category tag and label it with the correct tool_id.

Datatype Builder

The Datatype Builder has a similar function to the Tool Builder. The design of this module is still in progress. All of the ISIS[h] filters work with ISIS[h] cube formatted images, and there are ISIS[h] programs that convert different types of images to these cubes. The system will be designed in one of two ways. It will either be designed to create only ISIS[h] cube data objects and whenever a different image type is selected it will automatically convert them to cubes, or the system will create galaxy[f] data types for every possible type of image a user might want to work with.

If everything will be converted to cubes there will be no need to create a Datatype Builder. If there is a need to create data objects for every image type a Datatype Builder will be designed like the Tool Builder that edits the correct files and scripts. It will look similar to Figure 5, seen below.

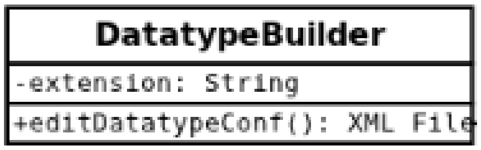


Figure 5: DatatypeBuilder UML

Galaxy Workflow Interpreter

The workflow interpreter (Figure 6) is a key piece of functionality and a big reason for choosing to use Galaxy[f]. It is responsible for guaranteeing each process in the workflow is passed correct parameters. Once the workflow is error free the interpreter runs the correct commands and handles passing the inputs and the outputs correctly from one command to the other.

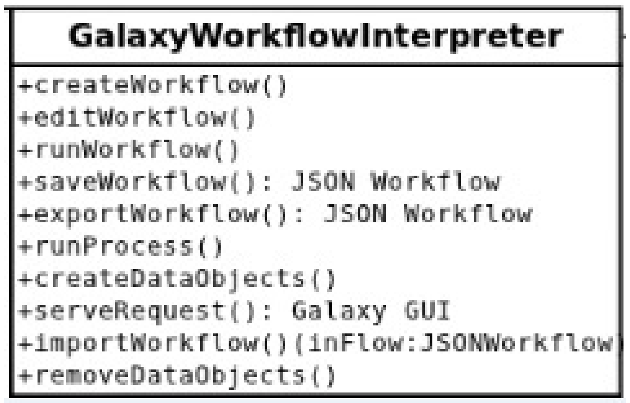

GalaxyWorkflowInterpreter
+createWorkflow()
+editWorkflow()
+runWorkflow()
+saveWorkflow(): JSON Workflow
+exportWorkflow(): JSON Workflow
+runProcess()
+createDataObjects()
+serveRequest(): Galaxy GUI
+importWorkflow()(inFlow:JSONWorkflow)
+removeDataObjects()

Figure 6: GalaxyWorkflowInterpreter UML

Ideally no changes will need to be made to the python[s] scripts that handle these functions. However, there is currently no full access to the source code.

Galaxy[f] GUI[g]

The web based GUI[g] Galaxy[f] uses is the other very useful feature for the USAT[o] project. By making the ISIS[h] interface web based users will be able to run filters or full workflows remotely. The Galaxy[f] framework[e] (Figure 7) already has tabs for sharing workflows, viewing/editing workflows, help section, and history/process analysis.

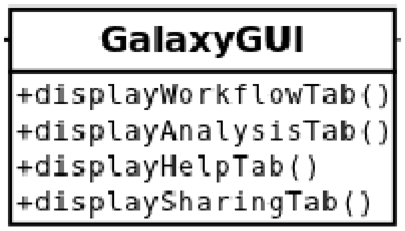


Figure 7: GalaxyGUI UML

Most of the current functionality is built using JQuery and the USAT[o] team plans on leave some parts, like the workflow viewing and editing, untouched. USAT[o] will be customizing many other parts of the GUI[g]. A new help section will be created. The team will also work to implement an embedded image viewer that will allow users to view the images without having to download the very large image files.

Workflow

The workflow tab of the GUI[g] is built using jQuery. It is a drag and drop environment where users can create data object nodes and filter nodes. Then they can draw connections between the nodes indicating inputs and outputs. The left pane contains a list of the filters and available data organized in categories that can be easily searched. The center pane is the blank place for organizing the workflows. The rightmost pane is where the user edits the parameters of the selected node. Menus are included for saving workflows, opening saved workflows, and exporting workflows.

Share

The share tab lets the user select saved workflows and shares them with other specific users, or makes a workflow public for anyone using the same server to use. If they wish to share a workflow with someone who is not using the same server instance they can export the workflow using the export functionality in the menu on the work-flow tab.

Help

Each filter will have a small help section that will be displayed when the parameters are being edited. The Help tab will include copies of those sections along with information about how to use the Galaxy[f] GUI[g]. It will also include instructions on how to install a Galaxy[f] server instance locally.

Analyze

The Analyze tab is where users will be able to run single filters at a time. Like the workflow tab, the pane on the left side will hold the list of tools. Unlike the workflow though the middle section will be for editing parameters, and the right hand pane will have a history of all the filters that have been run and on what data objects. The history will be like a linear workflow built one filter at a time rather than all at once and can be saved as a workflow if the user decides that might be useful.

Viewer

The viewer tab will be a place for users to view the images they are working with. The USAT[o] team is still researching how this will be implemented. One possibility is an embedded web page that links to sites ISIS[h] users already work with. The other possibility is a custom image viewer built by the USAT[o] team.

Image Viewer

The Image Viewer (Figure 8) will be located in the viewer tab. It will allow users to view the images they are working with as they are working with them.

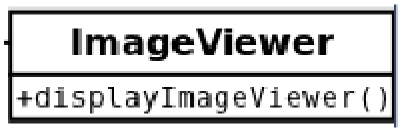


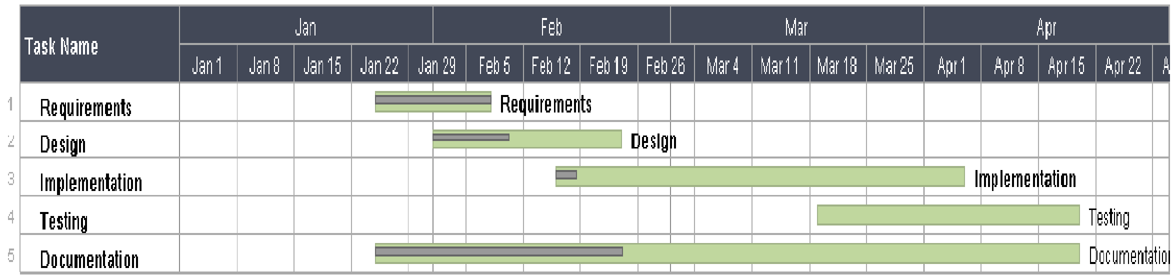
Figure 8: ImageViewer UML

Exact implementation is still being researched. The best solution would be a custom ISIS[h] cube viewer, but there are potential issues with the large size of the images. Another possible solution would be including an embedded web page that displays web sites ISIS[h] users currently use to download the images they want to work with.

User Databases

The user database holds user names, passwords, and saved workflows. It will be implemented in MySQL. Galaxy[f] already generates the tables it needs and such functionality will not be tampered with for the time being. The USAT team may integrate authentication with USGS's[p] Active Directory system before deployment.

Implementation Plan



Requirements have been examined and completed. As these are the guidelines for the entire project, reexamination of requirements will be necessary and conducted during important milestones.

Design is the main focus for this project. The chart above shows design ending when the Capstone document is due, however design will be revisited. Testing, meeting with the clients, and discussions during group meetings on the project will determine how design will effected during these revisits.

Some implementation has already been completed, though most of it has been more proof-of-concept programming. March will be when the majority of implementation is done, with a completion date within the first two weeks of April.

The plan for testing will hinge mostly on the availability of current ISIS[h] users. The USAT[o] team can test for bugs in the system, but user testing[p] will be of more importance to the success of the GUI[g].

Documentation will also be addressed throughout the development of the project. This will mainly be artifacts for the Capstone class, but also to practice coding standards to help communicate within the team and for future management. Documentation of all of the ISIS[h] applications is also a major part of the GUI[g].

Glossary

[a]Centralized: one interface that combines all other interfaces.

[b]Client-Server: a computer model where tasks and workloads are partitioned between a provider (server) and requester (client).

[c]Ease of Use: the quality of the user experience through the entire GUI[g].

[d]Expandability: the ability to accommodate additions to the programs capacity or capabilities.

[e]Framework: a reusable set of classes (or, sections of code) for a software system.

[f]Galaxy: an open source, web-based program that provides a scientific workflow, data integration, and data analysis persistence and publishing platform.

[g]GUI: provides the ability to interact with a computer using pictures and symbols, rather than having to memorize many complicated commands and type them in exact form.

[h]ISIS: Integrated Software for Imagers and Spectrometers; an image processing software package.

[i]Maintainability: the ability of a computer program to retain its original form, and to be restored in that form in the event of a failure or error.

[j]NASA: National Aeronautics and Space Administration.

[k]Reliability: the probably of a failure-free software option for a specified period of time.

[l]Robustness: a program that performs well under ordinary but also unusual conditions.

[m]Understandability: the ability for which a human reader can understand each module without having previous knowledge.

[n]Unit Testing: individual units of code/functionality is tested separately from the whole system.

[o]USAT: User System of Astrogeology Technologies

[p]User Testing: tests that involve having an actual user test the system.

[q]USGS: United States Geological Survey

[r]XML: a programming language with a set of rules for encoding documents. The difference between HTML and XML is that XML was designed to transport and store data, and HTML was designed to display data, with the focus on how the data is shown.

[s]Python: an interpreted, interactive, object-oriented, extensible programming language.