Design Document

The Disease Outbreaks Team

Abdulaziz Alhawas

Jean-Paul Labadie

Jordan Marshall

Luis Valenzuela

Introduction

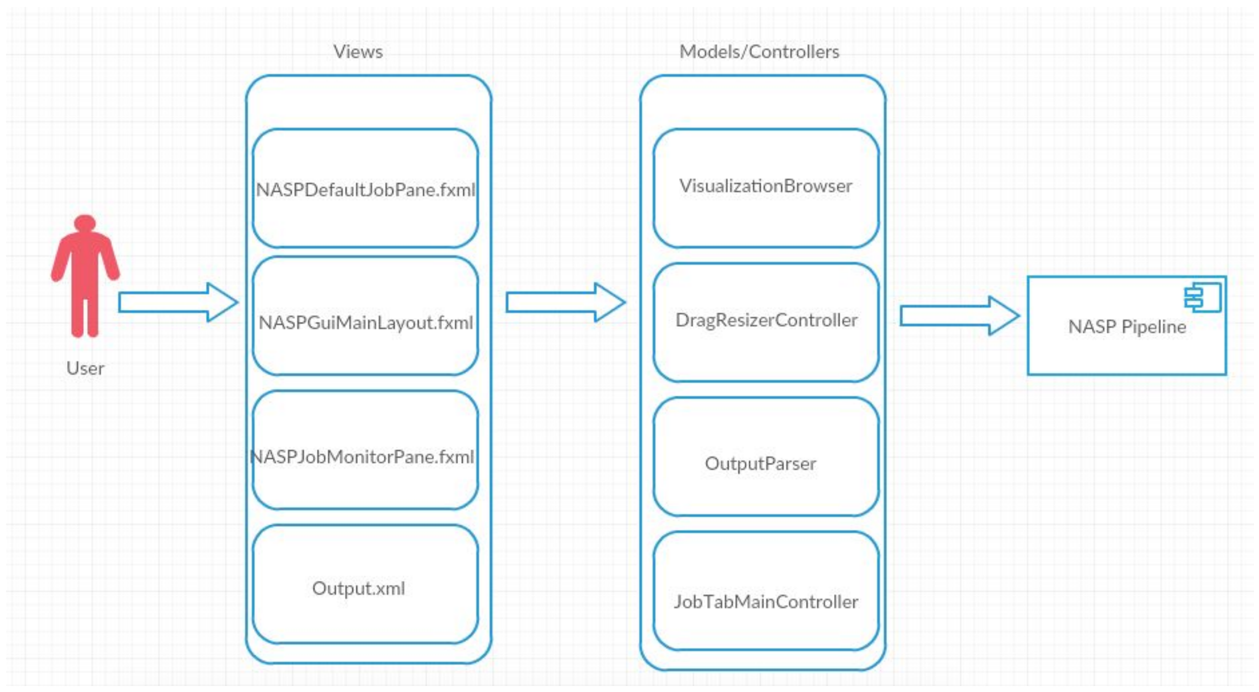
The team that we are working with at Tgen has tasks that require a substantial amount of computing power to complete. Due to that, the tasks are run on a remote computing cluster. Tgen has developed a tool named NASP that lives on those clusters and performs the needed calculation on files that Tgen produces. In order to run, NASP needs certain parameters which the biologists provide.

Our ultimate goal for this project is to create an interface for Tgen that takes user input, generates a correctly formatted XML, and sends that XML file to a computing cluster. Tgen would also like to have some sort of visualization of the data returned by the cluster. Currently, Tgen has a solution that is not ideal for the biologists interacting with the tool. Since the interaction with the tool is done entirely through the command line and not many people at Tgen are familiar with it, they have requested a tool with more user friendly interface.

The majority of this project will be completed using Java, JavaFX, and D3.js. JavaFX is being used to created the interface and we are using Java to implement the logic and communication with the computing clusters. We plan on using D3.js to implement to generate visualizations of the data in the form of a tree.

Providing this tool to Tgen would save them time and frustration of having to deal with a command line tool. Having an interface to interact will make the tool easier to use for users regardless of their familiarity with the command line.

Architectural Overview

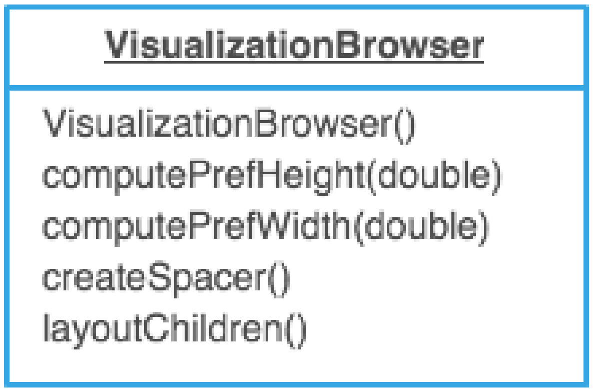


The solution has MVC architecture; views that hold most of the GUI which the user interacts with , models which has most of the system's logic that process the arguments passed by the views, controllers that handles the communication between views and models in the system, and NASP pipeline which is an external component that the system interacts with.

The system has four major views; NASPDefaulJobPane which holds the arguments that the user inputs to start a job, NASPGUIMainLayout that implements the toolbar which enables users to start jobs, NASPJobMonitorPane which allows the user to track jobs progress status, and Output which will provide the user with data visualisation.

The system also consist of models and controllers which has four important components; VisualizationBrowser which is the logic behind Output view, DragResizerController which improves the quality of the GUI, OutputParser which handles the output that get received from NASP Pipeline and parse it for visualization purposes, and JobTabMainController which handles all the user navigation that happens on the job tab. The Solution also interacts with NASP Pipeline and provides it with xml scheme that allows it to start the job.

Module and Interface Descriptions

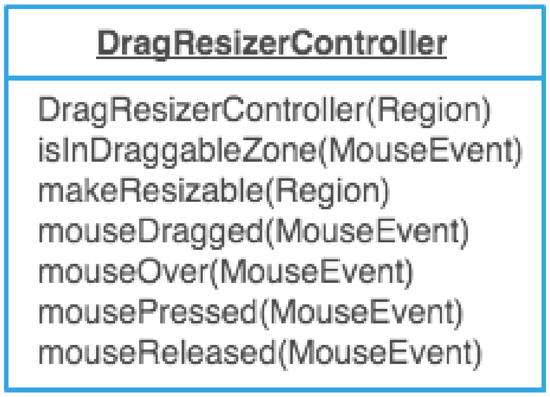


VisualizationBrowser

VisualizationBrowser class is an important class for creating the visuals that the user wants after receiving their output data in the GUI. This is important to tGen so that they can create tables and phylogenetic trees after running a job to more easily compare their results and data

- **VisualizationBrowser() :** method will start by loading a web page, adding the page to the scene, and also applying the styles to the scene.
- **computePrefHeight() :** and computePrefWidth() will create the proper size height and width respectively, to allow the browser to be viewed properly inside the scene of the GUI. This will be done by taking a percentage of the parent window and returning a proper window ratio for the monitor.

- **createSpacer() :** is a required method by Java to control the size of the Browser that has currently been created and showing, this is essentially where every instance of the Browser lives.
- **layoutChildren() :** method is needed to get the height and width of the browser and to set the proper dimensions of the child elements that are built inside of the browser component.

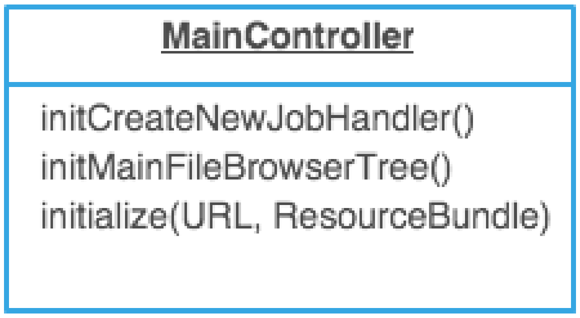


DragResizerController

DragResizerController is what we will use to add mouse listeners to a region and make it resizable by the user by clicking and dragging the border in the same way as a window.

- **DragResizerController() :** Constructor that creates and instantiates the region that we will be clicked and dragged for resizing.

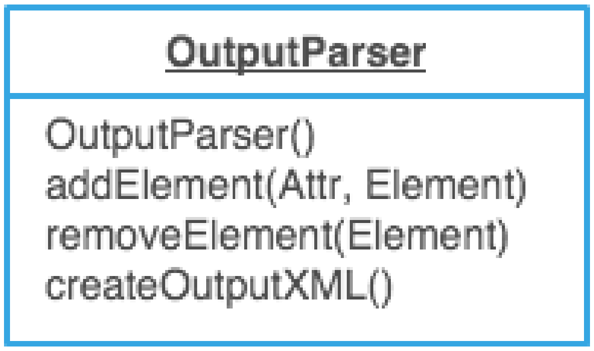
- **isInDraggableZone() :** is called on a mouse event, and then checks to see if that mouse click is inside a valid zone. It essentially checks to see if the thing that is trying to be dragged is in fact draggable.
- **makeResizable() :** is invoked after we validate that a zone is draggable and then creates an instance listening to the mouse drags to make the new size of the region.
- **mouseDragged() :** listens for a mouse event and returns a x_axis variable to which the new window size will be adjusted to.
- **mouseOver() :** checks to see if we are dragging in a draggable zone and then sets the cursor to the new resize value.
- **mosuePressed() :** is responsible for ignoring clicks outside of a draggable event.
- **mouseReleased() :** resets the cursor to the default state when released.


MainController
initCreateNewJobHandler()
initMainFileBrowserTree()
initialize(URL, ResourceBundle)

MainController

The MainController Class monitors and responds to changes at the highest level of the GUI. It manages the three main Panes of the GUI and the main menu bar.

- **initCreateNewJobHandler() :** On startup, creates a Handler which monitors the "Create New Job" button in the main menu. When the "Create New Job" button in the main menu is pressed, this Handler will add a new Tab to the JobTabPane with its own Handler.
- **initMainFileBrowserTree() :** On startup, creates a Tree which visualizes the user's file system, and displays this tree in the file browser pane. This Tree allows users to drag and drop their selected files or directories into containers in a JobTabPane.
- **initialize(URL, Resource Bundle) :** On startup



OutputParser

The OutputParser class is used just prior to job initialization. Once all required fields in the current JobTabPane have been populated, the user may click "create new job" or "save job as template."

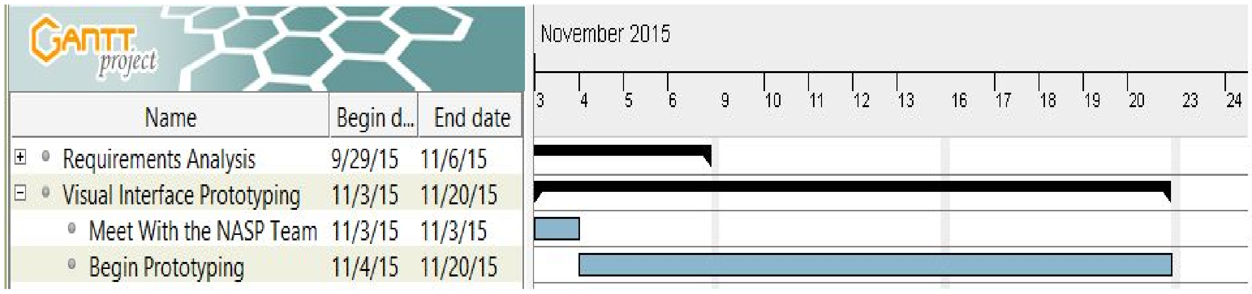
If the user selects "create new job," the OutputParser is used to add the current state of the JobTabPane to the XML model. This is a multi-step process. As fields are populated in the JobTabPane, Elements and Attributes are added to the Document (which is stored on the Java Stack). When the user clicks “create new job” the Document is written to disk.

- **addElement(Element element, Attr attribute) :** creates a new Element Node in the XML DOM with the supplied inputs. Note that this is represented in memory, but not yet written to the disk. Elements and Attributes are defined as Classes based on the NASPInputSchema.xsd
- **removeElement(Element element, Attr attribute) :** removes a previously added Element from the XML DOM if it exists. Note that this is represented in memory, but not yet written to the disk
- **createOutputXML(String filename) :** writes the current XML DOM to the disk using the filename given in the default output directory. This finished XML represents the entirety of the NASP tool job request, and can be sent to a remote service running NASP to begin a new job

Implementation Plan

Interface Prototyping

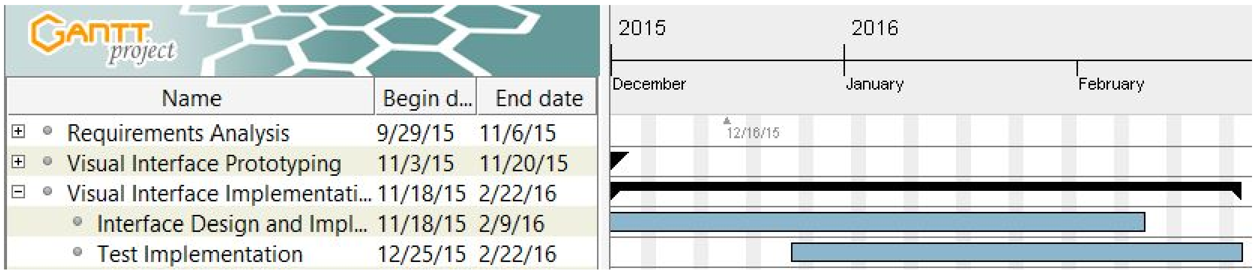
The implementation for this project began with meeting with the NASP team at Tgen and researching what types of tools they were already using and what design aspects of each of those tools they liked and disliked. At this point we began creating some prototypes of the interface taking this feedback into consideration. After the layout of the interface met the requirements the implementation of it began.



Name	Begin d...	End date
Requirements Analysis	9/29/15	11/6/15
Visual Interface Prototyping	11/3/15	11/20/15
Meet With the NASP Team	11/3/15	11/3/15
Begin Prototyping	11/4/15	11/20/15

Interface Implementation

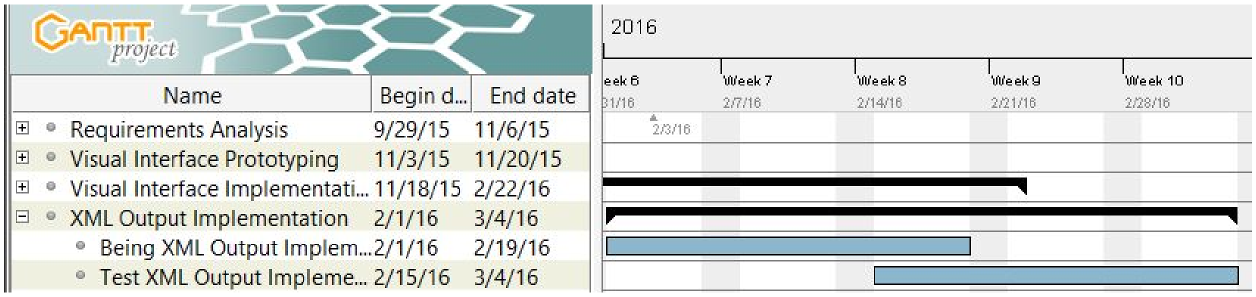
The layout of the interface at this point should be nearing completion there we may still be moving aspects of the interface around. The bulk of the work at this stage is at the backend. Panes, text fields, buttons, etc. should be identified properly within the code and the implementation of the logic needed to fulfill the functionality should be underway. This stage of the project will overlap with the XML output implementation which will start once the relevant aspects of the interface have been properly identified and implemented.



Name	Begin d...	End date
Requirements Analysis	9/29/15	11/6/15
Visual Interface Prototyping	11/3/15	11/20/15
Visual Interface Implementati...	11/18/15	2/22/16
Interface Design and Impl...	11/18/15	2/9/16
Test Implementation	12/25/15	2/22/16

XML Output Implementation

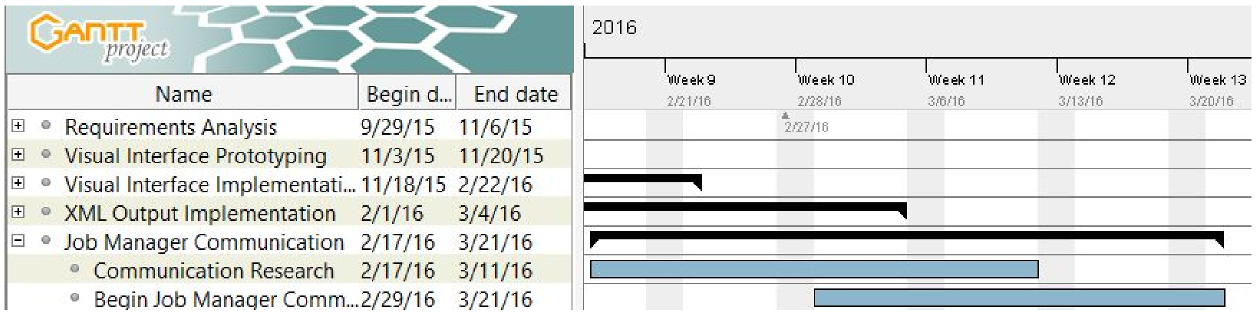
As stated before this functionality should start being implemented once the proper fields within the interface have been identified. Since this is the file that NASP takes as input it need to be properly formatted before being sent off. The implementation of the output requires us to be familiar with the XML schema that Tgen's command line tool generates. Once the XML is generated by our tool we will need to develop a way to test the integrity of the file.



Name	Begin d...	End date
Requirements Analysis	9/29/15	11/6/15
Visual Interface Prototyping	11/3/15	11/20/15
Visual Interface Implementati...	11/18/15	2/22/16
XML Output Implementation	2/1/16	3/4/16
Being XML Output Implem...	2/1/16	2/19/16
Test XML Output Impleme...	2/15/16	3/4/16

Job Manager Communication

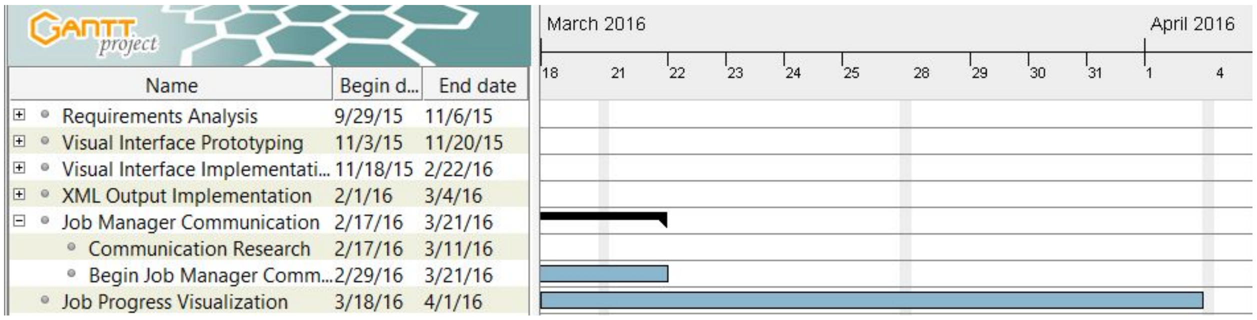
The next step in the process is to develop a means to send the generated XML file to the computing cluster where NASP lives. Given that Tgens current tool already has a means of achieving this it may just be a case of porting over the code to our tool but we are prepared to create our own implementation if it results being more difficult than that. To achieve communication we will have to do some researching both within their tool and outside networking resources.



Name	Begin d...	End date
Requirements Analysis	9/29/15	11/6/15
Visual Interface Prototyping	11/3/15	11/20/15
Visual Interface Implementati...	11/18/15	2/22/16
XML Output Implementation	2/1/16	3/4/16
Job Manager Communication	2/17/16	3/21/16
Communication Research	2/17/16	3/11/16
Begin Job Manager Comm...	2/29/16	3/21/16

Job Progression

Since jobs sent to NASP can take a considerable amount of time to complete Tgen would like to have a way to check on a job's status. We will need to look into whether the computing clusters have a way to easily check on jobs that are running. Once we have an understanding about how the clusters handle this we will start implementing a way to poll them for progress and display that result to the user in the form of something like a progress bar or a percentage.



Name	Begin d...	End date
Requirements Analysis	9/29/15	11/6/15
Visual Interface Prototyping	11/3/15	11/20/15
Visual Interface Implementati...	11/18/15	2/22/16
XML Output Implementation	2/1/16	3/4/16
Job Manager Communication	2/17/16	3/21/16
Communication Research	2/17/16	3/11/16
Begin Job Manager Comm...	2/29/16	3/21/16
Job Progress Visualization	3/18/16	4/1/16