Final Report

The Disease Outbreaks Team

Abdulaziz Alhawas

Jean-Paul Labadie

Jordan Marshall

Luis Valenzuela

Introduction
Process Overview
    Interface Prototyping
    Interface Implementation
    XML Output Implementation
    Job Manager Communication
    Job Progression
Requirements
    Functional
    Non-Functional Requirements
Architecture
Testing
    Unit Testing
        Job Saving/Loading Module
        Methods and Unit Tests
        User Settings
        Methods and Unit Tests
        Network Module
        Methods and Unit Tests
    Integration Testing
    Usability Testing

Introduction

The sponsor for this project is Darrin Lemmer at TGen North. The strategic plan for TGen North is focused on diagnostic, analytic, forensic and epidemiologic research related to pathogens important to medicine, public health and biodefense. The research capabilities at TGen include a variety of DNA sequencing and PCR-based analyses, forensic analysis of outbreak and bio threat incidents, as well as advanced bioinformatics and computing infrastructure.

The Northern Arizona SNP Pipeline is the primary tool used by the disease outbreak division at TGen, the sponsor, to investigate fungal and bacterial genomes for the purpose of tracking disease outbreaks. The pipeline is optimized to run on a computing cluster, which can be remotely accessed. Currently, users interact with a command-line program, which emulates a 'wizard' style interface, asking the user for the locations of various files, parameters that need to be included with the data, as well as other options to include while running the process. This command-line program then generates a formatted XML document, after which is passed on to the computing cluster. The NASP tool uses the information in the XML to run the process.

This command-line tool, while effective, can be daunting and unforgiving for a non-programmer to use. Right now there is no way to clear an entry, adjust an entered parameter, or fix a mistake without resetting the whole command-line tool. The command-line tool does not allow the input of custom options, unless the XML document is directly manipulated, this is not feasible for users unfamiliar with XML and its structure. The users seek a way to pass in their data to the computer cluster, run NASP, and receive their output without having to go through the command-line or directly into the XML document. The primary objective of this project is to create a more user friendly graphical interface in which users of the current command-line tool will have more flexibility and increased ease of use.

The main focus for the GUI is to replace the command-line tool entirely, providing a robust and modular interface which improves the user experience. This goal is well within our abilities, as the command line tool is sufficiently abstracted from the NASP job process. Because the NASP pipeline is dependent solely on the XML, the GUI will be completely independent of the other technologies used in NASP. The core requirement of the project is then to provide a user interface which can generate this XML and pass it to NASP pipeline to begin a job. Features of the project will build upon this, including providing a visual representation of job completion by interacting with the job manager, and visualizing the phylogenetic trees which result from the job once it has finished.

Potentially, this GUI could set the standard for genetic SNP's phylogeny. Since command-line tools are the current route to manage the NASP tool currently, a graphical user interface will provide massive amounts of extra usability to users that are not only uncomfortable with the command-line, but also unfamiliar with the generated output that will be passed to the cluster.

Process Overview

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.

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.

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.

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.

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.

Requirements

Functional

The final implementation must be able to fully replace the command-line tool that is currently used to interact with NASP, while leveraging the advantages offered by a more advanced interface. To this end, it must provide an intuitive Graphical User Interface (GUI) which allows users to build a new job for the NASP tool. This GUI must also allow this job to be sent to a number of remote computing centers (currently ASPEN and MONSOON), or run locally.

To begin the process of creating a job in the new tool, users must be able to specify where output files will be written. Users must be able to provide either local or remote reference FASTA files. Users must be able to toggle the option to ignore duplicated regions found in the reference files. The user must be able to select the job management system that will be used, as well as the option to run without a job manager. Likewise, the tool must support interactions with the PBS/Torque, SLURM, and SGE job management systems. If a job management system is chosen, the system

must allow users to specify a queue or partition to be used for all jobs, or use the default queue. The system should also allow users to add additional arguments to the job request.

The user should also be able to supply local or remote FASTA files from external genomes. If remote external genome FASTA's are supplied, the user should be able to specify advanced settings for the NUCmer tool.

The user should be able to supply local or remote read files if they wish. The user should be able to select which alignment tools they would like to use, including BWA, Novoalign, and SNAP. Alternatively, users should be able to provide pre-aligned files such as BAM. Depending on the choice of alignment tools and files specified, users should be able to choose only related additional options.

If the BWA aligner was selected, users should be able to select options such as running the BWA samp/se tool, and the BWA mem tool. Furthermore, they should be allowed to define any advanced settings related to the BWA tool and optional tools.

Users should also be allowed to run the Bowtie2 tool. Users should likewise be able to set advanced settings for the Bowtie2 tool.

If users chose to include Novalign as an aligner step, they should be allowed to enable and define extra Novalign settings. For instance, users should be able to specify an alternate Novalign version from the default. Users should be able to supply additional arguments. Users should also be able to Novalign's runtime settings, such as defining queue and partition, maximum memory allotment, maximum CPU allotment, and maximum runtime allotment.

If users choose to run SNAP, they should be allowed to specify local or remote prealigned SAM or BAM files to be included. Users must also select one or more SNP caller functions, GATK, SolSNP, VarScan, and SAMtools. Users must also be allowed to provide local or remote VCF files if they are available.

If users elect to include the GATK SNP caller function, they should be allowed to define advanced options for the tool. If users elect to define advanced options for GATK, they should be able to: choose an alternate version of GATK, define the queue or partition for GATK to run on, define the maximum memory allocation for GATK, define the maximum CPUs allotted to GATK, define a maximum runtime to GATK, and define additional arguments for the GATK runtime.

Users should also be allowed to define how NASP will filter results based on coverage. Users should be able to select a minimum coverage threshold, including zero (no coverage filtering). In addition, Users should be able to filter based on the proportion of reads that match the call made by the SNP caller. Users should be to enable this filtering, define the minimum acceptable proportion.

Users should be able to define advanced parameters for the MatrixGenerator processing step. These should include: define an alternative

MatrixGenerator version, pass additional arguments to the MatrixGenerator, define the queue or partition for the MatrixGenerator to run on, define the maximum memory to be used in processing, define the maximum CPU count to be used in processing, and define the maximum run time of the job in hours.

Finally, the user should be able to decide if the generated matrix should include all reference positions, or if the generated matrix should mask low-quality calls. Based on the user's settings, as defined above, the tool should then create an XML document which conforms to the schema defined by TGen North. This XML is the interface used by the NASP tool, and completely defines the job and tasks desired by the user.

The tool must also be able to connect to the remote computing centers and start the job, using the XML generated and providing it to the NASP pipeline through the job managers selected. Currently, the tool must support interactions with the job managers PBS/Torque, SLURM, and SGE. The tool must also provide an interface for the user to visually track the progress of started jobs via said job managers, and to retrieve the files generated by the job upon completion. Finally, the tool should provide the user with visualizations of the generated matrix, including diff-like side-by-side views of SNPs, and a phylogenetic tree. Additional features may expand upon these visualizations, and could allow users to draw upon and annotate trees, collect individual sample information, and additional filtering.

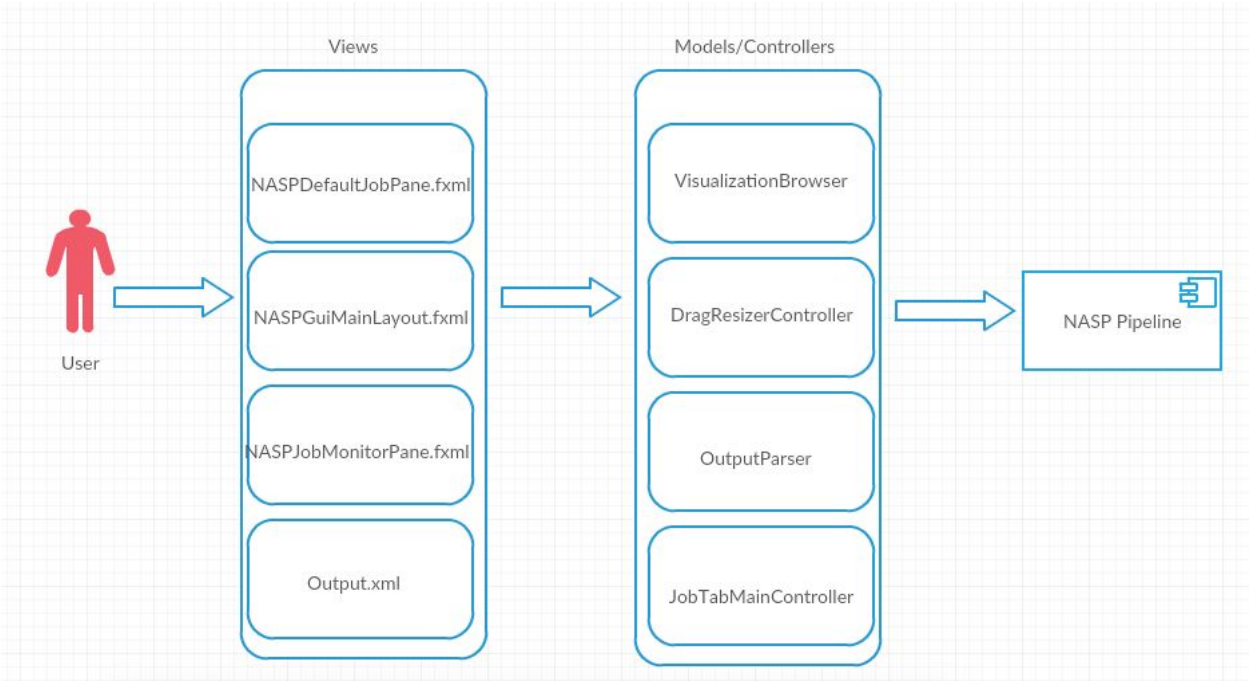
The tool should also be 'modular' where possible, but especially in terms of the visualizations, providing users a straight-forward method for expanding or changing functionality should the need arise after the project is completed.

Non-Functional Requirements

The GUI shall have a simple design that should not require users to undergo additional training. The GUI design needs to look similar to TGen's existing UI programs, and should follow their design patterns. Results must be processed and displayed to the GUI to allow users to visualize data in an organized way.

Additional tools should be easily integrated into the solution. The GUI needs to be implemented in a manner which produces reliable and correct XML documents. The GUI shall also provide a visual representation of NASP job completion status. Finally, the design shall be documented using TGen's design templates.

Architecture



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.

Testing

Unit Testing

The graphical user interface being implemented consists of a variety of parts that all interact with each other to provide useful functionality within the interface. Many of the modules that provide that functionality depend on the correctness of other modules. Because of this it is important to make sure that they are functioning as expecting and returning values that make sense and are correct for any given input.

Developing unit tests will ensure that our modules are correct and will help us identify and correct issues within the program. The following are important modules, their arguments, and a discussion of the unit tests that will validate their return values.

Job Saving/Loading Module

This module provides a time saving feature in the form of loading and saving job settings. It saves settings by taking the contents of the interface and putting them into a Java object named NaspInputData. The contents of this object are then saved into an XML file that is formatted in such a way that it can be sent and run on NASP without having to be processed further. The loading works in the opposite direction. First an instance of the NaspInputData object is created, the contents of the XML file are then saved into this object, and finally the interface is populated.
Methods

Methods and Unit Tests

jaxbXMLToObject

This method takes an XML file as argument and returns an instance of NaspInputData that contains all of the settings within the XML file.

jaxbXMLToObjectTest

This unit test passes an XML file to the jaxbXMLToObject method and stores the reference to the NaspInputData class that it returns. The values within this class are then compared to the

corresponding values within the XML file. This test passes if the values match and fails otherwise.

User Settings

The user settings module handles the saving and modifying of settings such as username, URL, port, and job manager. These settings are stored on initial login and later used to connect to a given cluster therefore it is important to make sure that the information is being stored and returns all of the fields correctly

Methods and Unit Tests

getCurrentRemoteSettings

This methods loads the current remote settings from the configs directly into a JSON object. It then accesses a key within this object corresponding to the remote settings and returns its contents in the form of a JSON object if the key exists otherwise it returns null.

getCurrentRemoteSettingsTest

The test for this will call getCurrentRemoteSettings, save the JSON object returned, and validate that the contents of the object to ensure that the returned objects contains the remote settings.

setCurrentRemote

This method simply sets a new setting with the key "Current Remote". Initially the contents of this key are empty.

setCurrentRemoteTest

The test for this method simply calls the setCurrentRemote then calls getCurrentRemoteSettings and ensure that there actually is a "Current Remote" key in the JSON file.

addRemoteSettings

This method actually sets the contents of a setting key within the JSON configuration file such as the Current Remote key.

addRemoteSettingsTest

Again this test will call the method to add the remote settings to a key and then verify that the contents of that key are correctly set.

removeRemoteSettings

This method takes the name of a setting as an argument. It then uses this name to search the JSON configuration object and remove the corresponding settings.

removeRemoteSettingsTest
: This test calls the method on a JSON file and then makes sure that the setting is removed by trying get the contents of the key removed. This should return null.

readSettings
: This method takes the path to a file as an argument and returns a JSON file with the contents of that file.

readSettingsTest
: This test calls readSettings and compares the contents of the returned JSON object with another JSON object that should match.

writeSettings
: This method takes a path and a JSON object as arguments. It then writes the content of that JSON object into the path specified.

writeSettingsTest
: The test for this calls writeSettings and then calls readSettings and verifies that the contents of the JSON file are correct.

getCurrentServerURL
: This method loads the current remote settings into a JSON object and returns the contents of the URL key.

getCurrentServerURLTest
: This test will call the method and verify that the URL returned matches the expected URL.

Network Module

The network module handles all of the communication with the clusters. This modules role within the overall program is important as its the bridge between the interface and the NASP tool. It needs to be able to form a connection, send commands, and transfer files from the cluster to the local machine and vice versa. The following tests will ensure that the methods that this module is comprised of function correctly.

Methods and Unit Tests

intiSession

This method takes username, password, url, and port as arguments. It then simply creates an instance of the Session class.

initSessionTest

The test for this calls initSession on an instance of the NetworkManager class, calls the initSession method of the class, then calls the getSession method, and checks if the returned value is null or not. If it is null the test fails.

openSession

This method uses the session instance that initSession creates to form a connection using the arguments passed to in.

openSessionTest

This test passes if the call to openSession does not result in an exception. Open session will attempt to create a connection will throw an exception is it fails.

closeSession

This method closes all channels that are currently open and terminates the connection.

closeSessionTest

This test calls closeSession on an open session then calls isConnected on the session. If this returns null the test passes.

upload

The upload method takes a file and an absolute path as arguments. It then uses the connection to the cluster to transfer the file to the cluster into the specified path.

uploadTest

This test calls upload and then checks for the file in the directory on the cluster. It passes if the file exists.

download

This method takes the an absolute local path and an absolute remote path as arguments. It then copies the file from the absolute remote path to the absolute local path.

downloadTest

The test calls download and checks whether the specified filename exists in the local directory.

runNaspJob
: This method takes a path to an XML file as argument. It then uses the connection to the cluster to send commands and start a NASP job with the XML file. It then returns the ID of the job.

runNaspJobTest
: The test calls runNaspJob and passes if runNaspJob does not throw an exception and returns an job ID.

Integration Testing

Integration testing ensures that the separate components of the GUI communicate with each other properly. Once we have checked the functionality of each separate class during unit testing we combine them together and run them as a group, luckily our project involves building a GUI so this already puts everything together for us. Integration testing is used to secure the integration between the view of the GUI, the inputs, and the intended output.

We chose to start with a more top-down approach, in that we started at the very top of the hierarchy and worked our way down to individual branches from there. Specifically we used a breadth-first process, in this case we started with the overall integration of the classes, fxml, and GUI as a whole. Once we were unable to visually see any obvious problems of integration between components we started to dig deeper into our interface and start coupling out components that made logical sense to be tested and working with each other.

Conveniently for the team, we are able to do some level of integration testing every single time we run the GUI from our local machines. Each time the run button is clicked the classes such as the MainController, the VisualizationController, the OutputParser, and so on are packaged all together with the fxml which dictates the look and layout of the GUI. In this way integration testing has been fundamental in our iteration process by testing and refactoring the way things are laid out within the user interface itself. In this way the whole team has been able to contribute to the integration testing portion of our project.

Some of the key things that were looked out for throughout the integration of the project were specifically how each piece of the GUI responds with one another. For instance, does the main layout fxml page display the components of the main window in a rationally sound way. We

also need to know if a drop down list that is implemented throughout the fxml is grabbing the proper values, if any, from the main controller java class. Because we are fortunate to have explicit visual cues in our GUI we are able to visualize a problem between the integration of two or more components and go back and edit the source code accordingly.

The final steps of integration testing are currently underway and requires careful scrutinization in the way the user interface works with Tgen's network. Because our GUI is currently sound in terms of stand alone integration, the real tests start to show in the form of the ability to local and network drives at Tgen, the ability to remotely access their clusters to create and send jobs, as well as grabbing and visualizing feedback, among other unforeseen problems. This process will not be extremely different than how the GUI was integrated amongst itself on a local environment.

Usability Testing

Usability testing is essential for the success of GUI based systems. In order for a user interface application to be successful, end users should be considered throughout the designing process. For or system, we focussed on the end-users experiences and on what they were comfortable with when we were designing our beta prototype.

We had a meeting with our sponsor and a number of his colleagues that were intended to use our interface where we interviewed them. We sat down with them and took a look on what tools they were using currently. We discussed with them what parts of the interface they liked and which they didn't like and why. We took notes with the intention of designing our prototype to have similar look and feel with the tools they already comfortable with. This process will help us design an intuitive interface that our audience would be very familiar with and hopefully would have a slight learning curve.

After carefully reviewing the tools that the Tgen employers are comfortable with, we began establishing patterns between them and began understanding why these interfaces work very well with our users. Figure 1 shows an example of one of the tools that tgen employers use. This tools gave us an idea of the basic structure they were comfortable with. We discovered that accessibility is key and our end-users like interfaces that has a lot of functionalities in one place. Their type of work requires them

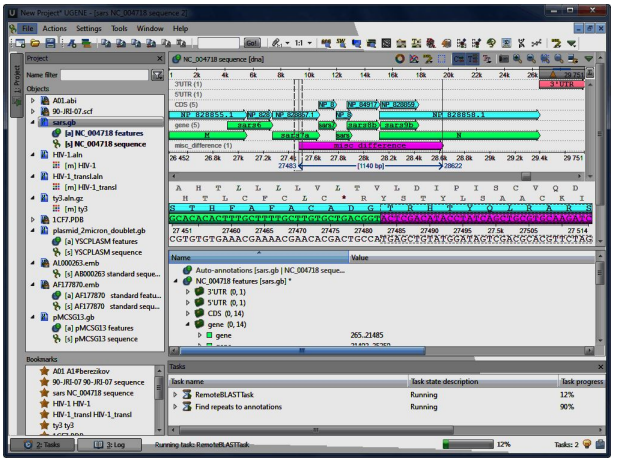


Figure 1. Ugene, a common informatics GUI

to use those tools frequently and for multiple tasks. Having multiple functionalities in one place enables them to save time and ease their work flow.

After we finished designing our beta prototype, we scheduled a meeting with our sponsor and a number of end-users. We presented our prototype to them and took notes of their initial impressions. We began modifying the design based on their feedback until they were satisfied. Including end-users into the design decisions made them familiar with the GUI and that would reflect very positively in the usability testing.

There are many techniques to test a usability of an interface. We are aware that the way we presented the prototype to the end-users may create biased opinions. Bias feedback is expected because the designers were in the same room with the end-users which may caused them to not say what they really think of the GUI and not give any negative feedback.

In order to solve this issue, we are thinking of having a two room testing environment where we design a lab manual and place the end-users in one room with the lab manual and record their interaction with the interface using video cameras. The designers will be in a separate room monitoring the testing process and taking notes without any interactions with the end-users. We believe that this process will reduce bias feedback significantly.

We will review the footage to try and discover where the users had problems with the interface and which aspects of the GUI they were having trouble with. This usability test will help us significantly to design an intuitive interface that Tgen and our sponsor would be very happy to use.