May 11, 2009

"Tree of Life" Teaching Tool
CIS 499 SENIOR PROJECT
FINAL DESIGN DOCUMENT

Madeline Yasner
Advisor: Val Tannen
University of Pennsylvania

PROJECT ABSTRACT

The Tree of Life Web Project is an endeavor to catalogue and classify data about "biodiversity and the evolutionary relationships of all organisms"[1]. Originally created for biologists, online projects like the Tree of Life have become popular to educators, students, and anyone searching for information about a specific group of organisms or about the diversity of life. One of the goals of the Tree of Life Web Project is to "aid learning about and appreciation of biological diversity and the evolutionary Tree of Life"[2]. It is within the scope of this goal that my project resides.

I have worked to create a software application that allows students to explore, both simply and thoroughly, the every-growing catalogue of all living things. This will involve the integration of phylogenetic (evolutionary) tree data with image data of living organisms, followed by the visualization of these two forms of data in helpful ways. Users will be able to correlate relevant data from both sources and interact with the program's visualization to edit or manipulate the data representation.

Project blog:
http://adventuresofmaddy.blogspot.com

1. INTRODUCTION

My project will use phylogenetic tree data stored in the TreeBASE database, combined with image data for living organisms, and enable students to begin exploring this vast amount of information in intuitive and constructive ways.

1.1. Significance of Problem or Production/Development Need

The incredible amount of data made available by TreeBASE is only as valuable as a user's ability to interact with it and learn from it. My goal was to lay the groundwork in creating a software tool that will allow students to make use of this great body of knowledge in a way that is most suited to their needs. The ability to individually explore

[1] "Goals of the Tree of Life," http://tolweb.org/tree/home.pages/goals.html
[2] ibid.

this vast store of information will greatly benefit students' understanding and appreciation in hopes of stimulating their interest in evolution and the sciences in general. In addition, visualization of such data will be helpful in understanding all forms of modern biological research, not only for students but also for anyone from amateurs to educators to the researchers themselves.

1.2. Technology

I decided to build upon the existing PhyloWidget (www.phylowidget.net) software. PhyloWidget is "a program for navigating and manipulating phylogenetic trees"[3]. The program is open-source and written in Java, which allows it to be as portable as possible. My own code was also written in Java, and I continued to use the Processing library used by PhyloWidget for the user interface.

I wrote my code in Java using Eclipse. Since the code for PhyloWidge is open source and available online, I needed a SVN client to download the latest version and update my download if there were to be any important changes made during the course of the semester. The SVN client I used was called Subclipse and worked within the Eclipse application.

The phylogenetic trees used come from TreeBASE (www.treebase.org), an online database storing these evolutionary trees, submitted by published biologists and researchers. It functions as a searchable online inventory of phylogenetic knowledge, which is always growing and changing with new research. The files I use to demonstrate my program all come from TreeBASE, and the eventual goal with this project is to have a program that can search TreeBASE and display a tree based on the user's search criteria. Currently, an updated version called TreeBASE II is under construction. My advisor, Val Tannen, is working on this project with William Piel, advisor for the PhyloWidget project, and others.

I accessed two online image databases to find images of the organisms in TreeBASE's phylogenetic trees. One of these is Morphbank (www.morphbank.net), a searchable database containing images of organisms for use by scientists. The other is NBII (http://life.nbii.gov), which describes itself as "a broad, collaborative program to provide increased access to data and information on the nation's biological resources"[4]. I also used the resources at UBIO (www.ubio.org) to search for common names or organisms by their taxonomic names.

---

[3] Gregory E. Jordan and William H. Piel, "PhyloWidget: web-based visualizations for the tree of life," *Bioinformatics* 2008 24: 1641-1642

[4] www.nbii.gov

1.3. Design Goals

My project will work to find a comprehensive way to interact with the data including both visual and textual information in a way that encourages exploration in a variety of directions while maintaining a simple interface structure for ease of navigation.

1.3.1 Target Audience.

Students and teachers of biology at many levels should be able to interact with this program and gain new exposure to evolutionary data.

1.3.2 User goals and objectives

The user should be able to interact with the program to search for images and information about desired living things, with the goal of seeing the visualization of the tree with this additional data.

1.3.3 Project features and functionality

The program should provide access to:
- Pictures of as many of the living things as are available.
- Scientific as well as common names.
- Diagrams of the selected organisms and their connections on the tree.

2. Prior Work

My work on this project is largely guided by previous programs for phylogenetic tree visualization and research about how programs like these fit into evolutionary educational programs. Of the existing programs, two of note are Tamara Munzner's TreeJuxtaposer and Gregory Jordan and William Piel's PhyloWidget. Denise Green and Rebecca Shapley researched the educational component.

The contribution made by Rebecca Shapley and Denise Green was a study interviewing biology teachers about their needs and producing a set of recommendations for educational biology software. They showed teachers a variety of trees visualization programs and asked for opinions about their benefits and shortcomings to determine what would make the best combination of features.

In general, they found that too much data can be overwhelming to students, so the first step is to break down that barrier by eliminating what they called the "overwhelm factor." Next they emphasized the importance of simplicity in visualizing a tree for the purpose of teaching. Finally they noted the importance of connecting new material with knowledge already familiar to students. A program that can display common names and pictures will hit anchor points for the students as they learn new material.

Of particular interest to me were the specific recommendations about the user interface and features of their ideal program. The first two listed below, images and

common names, are the ones that I chose to implement in my program. The others exist already in the program PhyloWidget, which I chose to build upon. I found Green and Shapley's recommendations encouraging, since the program I have been extending already implements many of these functions.

- Images of organisms
- Common name search
- Variable branch lengths
- Rank-free circular layout
- Animated changes
- Non-overlapping labels
- Resizeable window and text

Tamara Munzner is an associate professor at the University of British Columbia, and she researches information visualization, with specific work on phylogenetic trees. Her TreeJuxtaposer program deals with both visualization and structural comparison of trees, although the scope of my project is limited to the visualization aspect. (See Appendix: Figures 1 and 2) Of note are her accomplishments in the following areas:

- Scalability in Tree and Display Size
- Guaranteed Visibility of landmark nodes, regardless of user's navigation.
- Limited occlusion of other nodes due to labels

PhyloWidget is a program created for "viewing, editing, and publishing phylogenetic trees online"[5]. It was created by Gregory Jordan under the mentorship of Bill Piel, the Associate Director of Evolutionary Bioinformatics at Yale University's Peabody Museum of Natural History. It takes as input a tree formatted using the Newick tree format and generates an editable and customizable visualization of the tree. (See screenshots in Appendix: Figures 3, 4, 5, and 6).

Since PhyloWidget already implements many of the suggestions from Green and Shapley and is an open-source program, I chose to build the image and common name integration features into their program rather than starting from scratch. I knew that the amount of work to create a program that could implement Green and Shapley's many recommendations would be quite too large a task for a one-semester project, so by building on existing code I was able to focus specifically on the elements that PhyloWidget did not yet include. I should note, though, that the task of not only understanding but actually influencing someone else's code is a mountainous one. PhyloWidget has an extensive amount of code, and a lot of my time was spent on tasks like deciding where it would be best to interrupt one of their functions or store some variable. I think that my work in adding to someone else's code should certainly not be overlooked when it comes to judging the amount of work it has taken to implement what

[5] http://www.phylowidget.org/

might seem like small additions. I am very proud of my success not only in programming the features I chose but integrating them into the larger framework of PhyloWidget.

3. PROJECT DEVELOPMENT APPROACH

3.1. Algorithm Details

I have chosen to build on the existing PhyloWidget program. This will allow me to focus on combining images with trees and incorporating common names of organisms, without having to recreate a significant amount of code. I will take advantage of PhyloWidget's existing interface, tree rendering process, Newick parser, and many other features. Modifications and additions will serve the purpose of adding content fit for students and adapting the interface to be better suited for their needs.

The user begins by importing a Newick format tree like the ones attainable from TreeBASE. The program's overall algorithm is as follows:

1) Pre-parse the file to display taxonomic name labels

2) When prompted, search www.ubio.org for the common name

3) When prompted, search online image databases www.morphbank.com and life.nbii.gov for images of the organisms. Display those images alongside the leaf labels, and allow the user to select which image is displayed for each node, if more than one exists.

See overall flow-chart diagram for in Appendix: Figure 7.

3.1.1 Newick Pre-Parsing (Appendix: Figures 8 and 9)

In a Newick formatted tree file, each scientific name is assigned a number, and the string at the bottom with the parentheses encodes the structure of the tree, referring to each node by its number. Importing this directly into PhyloWidget yields a tree correctly formatted, but with numbers as the labels instead of the scientific names. My pre-paser interrupts PhyloWidget and reads in the file first. It replaces each number in the encoding with its corresponding name before sending the file to the PhyloWidget parser. So rather than outputting numbers, the program now outputs the names as the labels. This becomes very important later on when searching for common names, since the search can be done based on the label name rather than looking back to the original file.

3.1.2 Common Name Search (Appendix: Figures 8 and 9)

I interact with the website www.ubio.org, which searches based on keywords and can return the common name of an organism.

Once the program is signaled to search for common names, the UBIO search is called by each node and performs a search based on its label name. The program parses through the resulting page, which in the case of UBIO's website is a page listing the options found based on the search. It will select the option matching the search term, request that page, and parse through it to find the "common name" tag. The string for common name will be stored in the node, as a secondary label string, and at render time the user's preferences about which name to display will determine which label is drawn.

3.1.3 Image Search (Appendix: Figures 10 and 11)

Using the taxonomic name, I search the image databases of Morphbank and NBII for images of each organism in the tree.

Process of searching for and displaying images: User clicks "Search for Images" which calls both Morphbank and NBII searches with the label for each node. The searches connect to the website for each database, post request data to the search bar form, and obtain the resulting response page HTML.

Next the program parses through the response HTML looking for image links. If it finds them, they're stored as NHX annotations in the node calling the search. Before storage, the image passes through a check to make sure it is not a "No Image Available" image that displays when a link is broken or null. Later, during the render process, the ImageRenderer loads those URLs as image objects and displays them to the left of the label text for that node. In the case that multiple images are found for one organism, the first is displayed. The user can view the additional images and change which one is displayed on the tree by clicking inside the image.

3.1.4 Interface Integration

The image and common name are displayed on the tree leaf nodes. The user has access to the full range of control options available for manipulating the tree's display parameters that is already part of PhyloWidget. In addition, the user can control selecting the displayed image for each node, if more than one image is available. Finally, the user will specify whether the taxonomic name or the common name is displayed.

3.2. Target Platforms

3.2.1 Hardware

Because I will be programming in Java, I intend for my application to work on any machine that is capable of running Java-based programs or applets.

3.2.2 Software

Java
Eclipse
Subclipse (SVN Client)
Processing (UI Library, previously used in PhyloWidget)

4. WORK PLAN / PROJECT VERSIONS

4.1 Project Milestone Report A (Alpha Version)

Completed at the time of my Alpha review were the following:

Literature review: This was completed at the time of submission of my initial design document. See the section titled “Prior Work.”

Finalized decisions about programming language/platform, UI library: I will be using Processing, the UI library currently used by PhyloWidget. I will continue to develop in Java, using Eclipse. This will allow my program to be as portable as possible. Java also makes URL interactions quite simple, and since much of my algorithm involves searching online databases, this will be greatly to my advantage.

Became comfortable in Eclipse and reviewed Java programming language: Through a series of short tasks outlined below, I was able to learn my way around the Eclipse programming environment and refresh my memory of Java. I also learned the basics of URL connectivity, which will be quite useful in writing this program.

Finalized decision about building upon PhyloWidget: I have chosen to build on the existing PhyloWidget program. This will allow me to focus on combining images with trees and incorporating common names of organisms, without having to recreate a significant amount of code. I will take advantage of PhyloWidget’s existing interface, tree rendering process, Newick parser, and many other features. Modifications and additions will serve the purpose of adding content fit for students and adapting the interface to be better suited for their needs. For example, the interface with which a user can control the properties of an individual node is confusing and cumbersome. If time allows, I would like to modify this element to make node editing a simpler and more visually friendly process.

Familiarized myself with the existing PhyloWidget code: The code for PhyloWidget involves a series of packages, each with a variety of classes inside. Because the small test tasks I have done involved interacting with different

aspects of the code, I am now confident that I understand both the high level process involved in running the program as well as many of the smaller details. Of course I will encounter new things with every new part of the program, but I feel confident that I can understand the software and contribute to its modification.

Completed software design plan: I finalized the overall plan of how the program will be structured. This is described in detail in the "Algorithm Details" section of this document.

Testing projects for font/label interception: My first task interacting with the PhyloWidget code involved changing the font in which the tree label text is displayed. In the process I explored much of the source code and started to become familiar with what each section does. Because I was changing a font that was stored in a separate directory of my project, I also learned about how the file system hierarchy works for Java programs and where the program would need to look in order to find the file for a new font.
Next, I wrote a separate function to change the actual text of the label. This involved a great deal more familiarization with two particular areas of the code: the parser for Newick trees and the label renderer. Once I learned how the label text was stored during the parsing process, I was able to intercept the render call for getLabel() and replace it with a function to convert the label text to something else. In my function, I converted capital letters to their numerical equivalent (A=1) and replaced all other strings with "Hi Val!"

Programmed HTML search interaction: After learning about URL connectivity from the *Core Java 2* textbook, I wrote a program to test connectivity and interaction with the Morphbank.com search page. I was able to successfully connect to the website, post data into the search form, and retrieve the result. I filtered the resulting HTML to pick out the lines that contained image references, since I will need to do this when searching Morphbank for images of each organism. I wrote this as a separate program, but it can be easily integrated into PhyloWidget.

4.1.2 Project Milestone Report B (Final Version)

In addition, completed at the time of my final project submission are the following:

Gain access to image databases: Achieved with help from Greg Riccardi at Morphbank.net and a quickly attained understanding of the process of communicating with websites through Java.
Create mock-up of desired UI appearance:
My primary concern was to choose how to display images in a way that kept the page organized and uncluttered while allowing as large an image as possible.

The mock-up I created shows what I would recommend for future changes to the interface, in addition to the decision to locate the images on the left side of the label text so as to keep them in line.

Code image search through databases: As described in my algorithm section above, I interacted with morphbank.net and nbii.gov to query their image databases for images by posting the scientific name of the organism to the search form and parsing the resulting page to retrieve image URLs.

Code integration of images into the tree render process: I incorporated my code into PhyloWidget by storing the URLs as NHX annotations on each node, so that they could be handled during render time to display the images. In addition, since many of the successful searches yielded multiple images, I created an invisible UI Object using Processing to listen for mouse clicks and handle any click in the area of the node's image. Clicks in that area result in a call to the ImageSearcher2 object to retrieve the image from the next URL stored and display it in the original image's place. Once a user reaches the end of the available images, the same click will start the sequence over again.

Code UBIO search for common name: As described above, I posted search data to the ubio.org and parsed the resulting organism information page for the common name of that organism. I integrated this feature into PhyloWidget by adding a String variable for common name and changing the relevant functions so that the correct variable is returned to the render depending on which name the user has selected for display.

Make additional UI changes to improve functionality: I decided to update the organization of the toolbar options into categories based on how the user would interact with them. Some features are for tree layout, some affect the labels, and some the nodes overall. Other features impact the structure or format of the dree, so those were placed into a separate category. Whereas before, the toolbar links were created as the features were created and were thus organized similar to the code's structure, this new organization should be much more useful to anyone unfamiliar with the inner structuring of the code.

Code "No Image" test: I added a check function to filter out the "No Image Available" image that Morphbank occasionally outputs. The MorphbankSearch has a static variable storing the undesired image, and before an image URL is stored as an NHX annotation, the URL is passed to a verification function that compares first the height and width of the image and then each pixel with that of the "No Image" image, using a BufferedImage and the getRGB(x, y) function. If the image passes that test, then its URL is stored and the process continues as before.

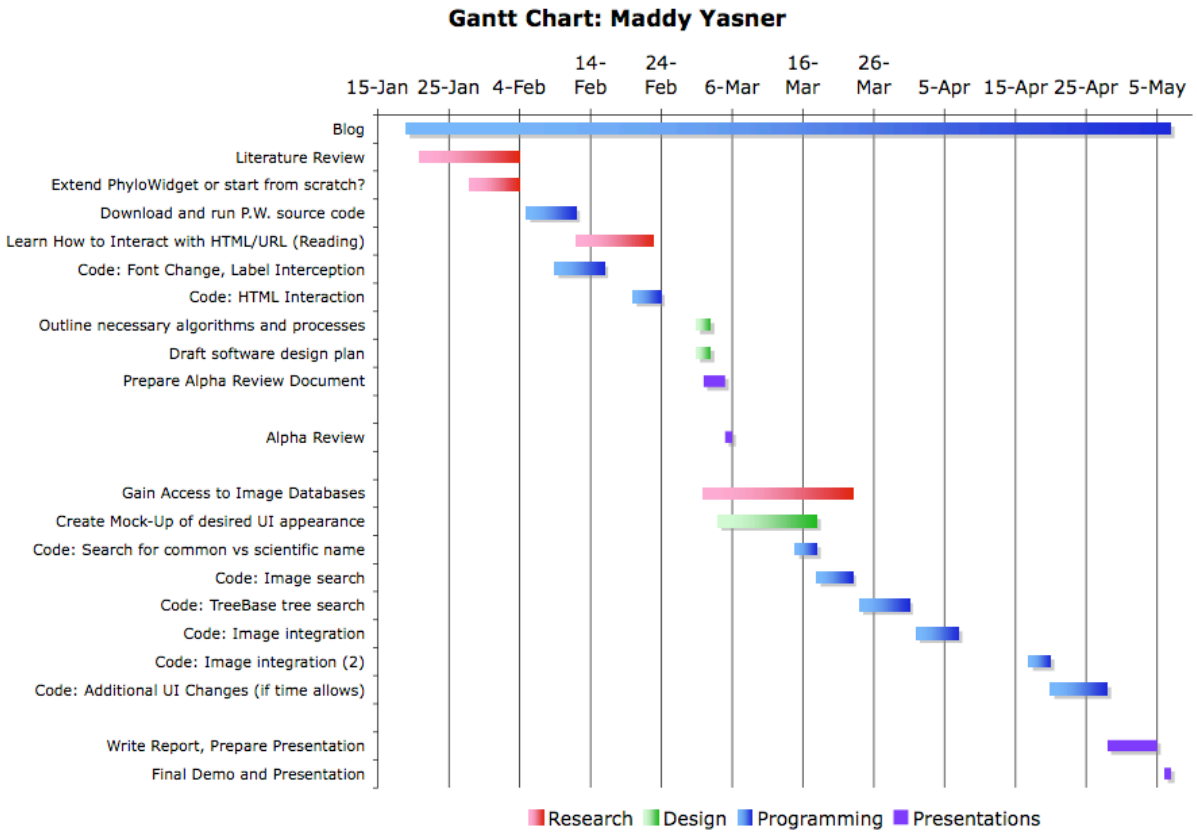
4.2 Project Final Deliverables

The “Tree of Life” Teaching Tool will be an extension of PhyloWidget that integrates images from online databases with trees in TreeBASE. The program takes as input a Newick tree file, pre-parses its text to display a tree using the encoded scientific names, and enables the user to search for images and common names for each leaf node.

4.3 Project timeline

- Complete Literature Review
- Choose whether to extend PhyloWidget or start from scratch
- Download and run PhyloWidget source code
- Learn how to interact with HTML/URL in Java
- Code a test function to change the font and label name of the nodes
- Outline necessary algorithms and processes
- Draft software design plan
- Alpha Review (first week of March)
- Gain access to image databases
- Create mock-up of desired UI appearance
- Code image search through databases
- Code integration of images into the tree render process
- Code UBIO search for common name
- Make any additional UI changes to improve functionality
- Code “No Image” test

4.4 Gantt Chart



5. RESULTS

5.1: Images – See Appendix

Figure 12: a, b, and c: Various zoom levels showing a tree with images loaded.
Figure 13: Updated toolbar organization
Figure 14: Tree with common names displayed

5.2: Demo Videos

"5.2.1.mov" Using my newly-integrated image and common name features, including image scrolling
"5.2.2.mov" Pre-existing PhyloWidget features and capabilities, including a tour of the updated toolbar hierarchy

6. FUTURE WORK

The work I have done is just the foundation, and a lot more can be done to make this program successful. A few of them are things I wish I'd been able to contribute myself.

The PhyloWidget interface is decent if you're a scientist but not if you're a high-school student. It could be a lot better, and DMD students are the perfect people to take on such a task, since it combines knowledge of the technical goings-on with an eye for design.

If a biology student wanted to learn about a specific organism, the best way would be if they could search for it by its common name and receive in return a tree with its evolutionary data, multiple images, and lots of other interesting information that might interest them. They should even be able to search for a list of organisms to get a tree about how they're all evolutionarily related. Implementing a TreeBASE II search in PhyloWidget would be a great addition. Instead of loading a file, they would search for one. That function alone expands the audience of this program beyond those who already have Newick trees to the common user.

7. REFERENCES

Hortsmann, Cay S. and Gary Cornell, *Core Java 2: Volume II – Advanced Features*, Palo Alto, CA: Sun Microsystems Press, 2002.

Jordan, Gregory E. and William H. Piel, "PhyloWidget: web-based visualizations for the tree of life," *Bioinformatics* 2008 24: 1641-1642.

Maddison, D. R. and K.-S. Schulz (eds.), *The Tree of Life Web Project*, 2007, Accessed on Feb 1, 2009 at http://tolweb.org

Munzner, Tamara, et. al. "Tree Juxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility," *Proc. SIGGRAPH 2003*, published as ACM Transactions on Graphics 22(3), pages 453—462. Accessed on Jan 30, 2009 at http://www.cs.ubc.ca/labs/imager/tr/2003/tj/tj.camready.pdf.

TreeBASE: A Database of Phylogenetic Knowledge, www.treebase.org, Accessed on March 3, 2009

8. WORK LOG

Week 1: January 18th – 24th

I met with my advisor, Val Tannen, to iron out some of the details of my project so that I can write a more complete design document. Our main focus is ensuring that my project will have a definite beginning and end and that it is a project I can successfully complete in one semester.
We reviewed some of my questions and went over the design doc template to make

some changes and additions to what I'd already started to piece together.

We decided that I need to do two things:

1) Read Tamara Munzner's paper on the Tree Juxtaposer to see how she visualizes trees and what I can learn from that. This will also get me started in the "Past Work" section of my design doc. I can choose to trace back from her references if I see things that I feel might be of additional use.

2) Spend a long while studying the already-written PhyloWidget code from Yale. I'll need to decide whether to start from scratch using that as a guide or whether to use it as a basis on which to build my code. Deciding how to utilize the PhyloWidget will have significant influence on the structure of my project, right down to the languages and libraries I'll be using to write the code.

In addition to that, I feel I'll need to do at least some background reading about Data Integration, since I'll need to do some amount of that to ensure that we're using fresh scientific data all the time. I'll also want to read up on algorithms and data structures that would be most useful for storing and manipulating data about trees.

Val is going to get in touch with his colleague at Yale who is excited about this project and seems willing to lend a hand, especially with the code from the PhyloWidget project that was created by one of his students.

At this point it's still quite too soon to detail things like algorithms of choice or even to attempt a Gantt chart. We're still very much in the "figuring things out" stage.

Week 2: January 25th – 31st

I completed reading the SIGGRAPH paper by Tamara Munzner about visualization and structural comparison of trees. I'm focused solely on the visualization aspect, for the Tree of Life dataset. Comparison of trees or subtrees is not within the intended functionality of my program. The paper was dense and would certainly require re-reading if I were to try to implement any of her algorithms, but reading it once gave me a sense of the topics she addressed and the difficulties she encountered.

Topics relevant to my project:
- Scalability in Tree and Display Size
- Guaranteed Visibility of landmark nodes, regardless of user's navigation. While her goal in doing this is for the sake of comparison, it would be of additional value to my program, where students may want to focus on the relationships between specific organisms.
- Occlusion of other nodes due to labels

Topics I don't need to bother with:
- Automatic Identification of Structural Differences between input trees
- Differences Characterization - exactly how two trees are structurally different

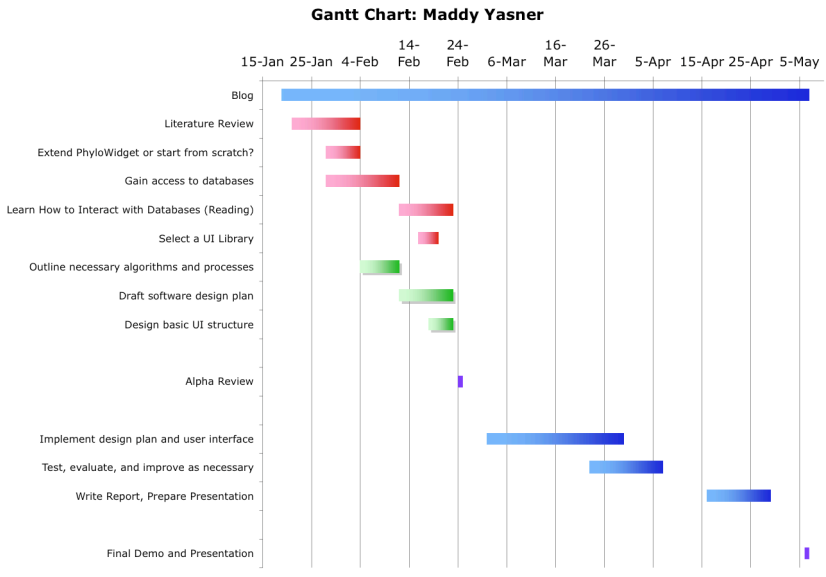
I met with Val again to continue finalizing my design doc and make a list of what I need to be focusing on next. What's left for the design doc is the background reading I still need to do about the Tree of Life project itself, both for the prior work section and the abstract.

Next Steps:
- download PhyloWidget standalone application, install it, and get used to how it works
- download Eclipse
- look back at old Java code I've written to re-familiarize myself with the language
- download PhyloWidget source code, open it as a new project in Eclipse, compile, and run it
- look for any code documentation on the PhyloWidget site

After That:
- experiment with small changes to the existing code to see how easily it can be manipulated
- decide then whether to build upon the PhyloWidget code or start my application from scratch

In addition, I'll also start brainstorming about the functionality I'd like to include in the program, so that we can start working towards a UI design as well as a software design. And, of course, I'll have a finished background section to add to my design document.

Original Gantt Chart



The chart will help me organize my workflow up through the Alpha presentation during the last week of February. From there on the chart is vague, since I know I'll arrive at a more detailed list of tasks as I proceed.

Week 3: February 1st – 7th

I had some difficulty determining which version of Eclipse to download and getting the source code from PhyloWidget to run as a new Eclipse project. I eventuall got Eclipse installed and used their “Basic Tutorial” for Java development to familiarize myself with the environment and its features.

Val helped me get in touch with Greg Jordan, author of the original PhyloWidget program. He recommended the two following site for both information and inspiration:

Rebecca Shapley's work on "Teaching with a Visual Tree of Life"
http://groups.ischool.berkeley.edu/TOL/

and the UK Wellcome Trust's "Wellcome Tree of Life"
http://www.wellcometreeoflife.org/interactive/

The second link looked really neat at first, and the graphics in the video are great, but it was tough to interact with and was pretty shallow in terms of the amount of information that was available.
The study done at Berkley may prove incredibly useful. The final report from their study was ~112 pages long, but I spent a while going through the powerpoint (with notes) from their presentation as well as reading briefly through the different sections of the paper to see what would come in handy later on. They have a whole section of recommendations based on interviews with teachers about what functionality they would like in a Tree of Life program. Just for a taste of some of their results, here are features that over 80% of respondents considered important or very important:

- Zooming in to any part of the tree
- Seeing areas of controversy
- Viewing the relationship of divergence events to geological time
- Seeing the distribution of important character states
- Bookmarking particular branches on the tree
- Accessing geographic distributions of groups of organisms
- Viewing the distribution of biological patterns across the tree

(Green and Shapley, p. 47)

I looked for Rebecca Shapley's contact information so that I might be able to get in touch with her directly, and although her e-mail address wasn't displayed on her website, I actually found her on Facebook and sent her a message. It'd be quite nice to have her direct input, since she conducted this massive amount of research that would be incredibly useful to making my project successful.

Week 4: February 8th – 14th

After getting some help from one of Val's graduate students, we were able to get it the source code running in an Eclipse project. I also had to download Processing as well as the com.lowagie library in order to get the code to run properly.

I also was unable to get the standalone version of PhyloWidget running, but once I got the source code working the standalone became unnecessary (although still troubling that it didn't work).

One of this week's goals has been to change some visual attribute of the PhyloWidget application. The purpose of this was the demonstrate whether the code was navigable enough to facilitate making changes, starting small of course. I found the file called FontLoader and was able to change the font from Vera (right) to Georgia (left) and Arial Black (bottom).

Sadly, changing to Wingdings or to a Hebrew didn't work, but I think that has more to do with the fonts themselves than anything in the program. It seems to only work with English fonts, which is fine. I was just experimenting.

Week 5: February 15th – 21st

I sifted through lots of code to find both the parser and renderer portions of PhyloWidget. The parser reads in the newick strings, and it is found in org.phylowidget.tree.TreeIO.java

```
public static RootedTree parseNewickString(RootedTree tree,
String s)
```

parses through all the notation, handles determining the different levels of the tree and the parent-child relationships.

When it's created a string of text it considers the label, it calls:

```
PhyloNode curNode = newNode(tree, curLabel, nhx, poorMans);
```

```
static PhyloNode newNode(RootedTree t, String s, boolean useNhx,
boolean poorMan)
```

This function deals with parsing NHX annotation, then

```
s = parseNexusLabel(s);
```

removes single quotes and replaces underscores with spaces, followed by

```
t.addVertex(v);
t.setLabel(v, s);
```

This allowed me to understand how labels are created and stored in tree nodes, which will be useful since I plan to be able to intercept those labels before render time and change them. In practice, the idea is to display common names in place of scientific names for the organisms in the tree.

Then I went on to find the renderer. The class LabelRender is found inside NodeRenderer.java. In the method render(), we have

```
canvas.text(tree.getLabel(n), offX - curTextSize / 3 - s,
offY - s - curTextSize / 3);
```

'n' is a PhyloNode and tree is a RootedTree so getLabel(vertex) is in RootedTree.java that calls

```
vertex.getLabel();
```

which is in PhyloNode.java (extends CachedVertex which extends DefaultVertex)

It is at this point that I intercept the label and change the return value so that the rendered label name is different than the stored one. I created a wrapper class for HashMap called NameLookup, which for testing purposes just stores mappings from each capital letter of the alphabet to the corresponding number, from 1 to 26.

I created a NameLookup object called map in PhyloNode.java and updated the getLabel() method to look up the value in the map and return the resulting string (in this case a number). Any label that is mapped to null in the NameLookup map is changed to the string "Hi Val". The replacement label string gets passed along all the way back to the render.

Another way I can do this is store the common names as NHX annotations on the tree, so that I don't have to constantly keep looking up the same names for each update render. I don't know much about NHX, but I'll figure it out if we decide that's the best way. Alternatively, I can just make a "common label" variable in PhyloNode objects and only do the lookup once, when the node is constructed.

Val contacted a friend who can help me get access to a website with a database of images so that I can integrate images of the organisms in to the display options.

Week 6: February 22nd – 28th

I created a program where I could test connecting to and interacting with a website. I tested my program using Morphbank, a searchable database containing images of organisms for use by scientists.

To get myself up to speed about Java and interacting with the internet, I read the networking chapter of "Core Java 2, Volume 2" Using their examples, I was able to create a program to connect with the Morphbank website and read in the HTML of the webpage I wanted. Next, since I'll actually want to input search data, I read through the HTML to find the necessary parameters and wrote a test function to post search data into the online form and return the result page. I filtered through the result HTML to find the lines containing image references and printed those to the screen. The next step will be harvesting those images and including them within the PhyloWidget program. I'll have to learn a little more before I understand exactly how to do that.

Greg, author of PhyloWidget, has been updating some of the old code. I wanted to find a way to work on the newest version without having to manually update each file myself based on his changes. The code is posted on GoogleCode, and in order to update my own local version Greg said I needed to use an SVN (Subversion) client. I found out that there's an open-source plug-in for Eclipse called Subclipse that does exactly what I need, and I was eventually able to download the latest code from Google and run the application.

Week 7: March 1st – 7th

I spoke with Val to outline the four specific tasks I'll need to complete in order to allow the user to input an organism's name and end up with a tree complete with images and both the common and scientific names for each organism. I divided up the major coding tasks and estimated how long it would take me to do each and which order it would make the most sense, and I used that information to revise my Gantt chart.

I've spent the rest of this week revising and updating my design document to reflect what I've accomplished and my goals for the rest of the semester.

I also contacted Anne Olsen at NBII to ask about how to search the NBII image database, at the recommendation of Greg Riccardi from Morphbank.

Week 8: March 15th -21st

See an illustration of my rough UI diagram in Appendix: Figure 15.

I began working on the portion of code that will take the scientific names of each organism in a given tree and search for the corresponding common name of each one. I am doing the search using the database at ubio.org. I can currently post a search term to the form and retrieve the resulting HTML page. I can parse that page to determine whether it is the page for a specific organism or whether the program has found multiple possible matches and is asking for more information.

The current issue is that the HTML I get back from the initial search is not the same as the HTML I get when I do the search in a browser and look at the resulting source code. That's definitely a sizeable issue. The URL is the same, but one's looking for more information and the other is an "Advanced Search" page that I can't even navigate to if I try in my browser.

I have written out my intended algorithm for parsing the HTML in order to find the common name of the organism.

Enter search term into ubio.org. Get resulting page

After text string "Scientific Match" search for "a href ='" string
Save the next text till "'" as a link string
Save a substring of the text after the next > and before <
Compare that to the search term.
If it's a match: follow the saved link

On the resulting page:
Find the text "name = 'Common name'"

After that point, find "namebankID"
Save a substring of the text after the next > and before <
Return the substring as the common name

Else: repeat by searching for next "a href = '" string

Stop when you've reached ".
Return either the search term or a null string as the common name.

Week 9: March 22nd – 28th

My computer died and needed a hard drive replacement. I was to be able to extract all of my files before handing it over to for repair, so in that regard I was quite lucky.

Week 10: March 29th – April 4th

I've got a brand new 500GB hard drive and reinstalled my operating system. I got Eclipse, Processing (UI), and Subclipse (SVN) installed so that I can run PhyloWidget and my own code as they were before the crash. So that was an accomplishment because I had to remember how I'd installed everything in the past couple months, but I got it working.

Today's issues are figuring out why my common name search is buggy and then proceeding onto dealing with searching multiple databases for images. At the same time I'll also need to make inroads into writing UI elements for each of these features and then integrating the code and the UI stuff into the PhyloWidget program.

I'm also concerned about the SVN process, because whenever I want to update to a new revision of PhyloWidget, Eclipse overwrites the one I'm currently using. If I make changes and start including my own code, I want to be able to download revisions without deleting my own work.

Week 11: April 5th – 11th

On Sunday I got sick, and after an evening in the hospital I spent the week pretty much in bed. (Diagnosis: it was just a virus.) Understanding that my time out would mean a little readjusting when it comes to procedures to reach my goals and deadlines originally outlined in my design document, I contacted Val and all of the other staff/faculty handling senior projects to alert them and discuss those changes and come up with a new timeline.

Week 12: April 12th – 18th

Sunday, April 12th:
I spent time reading up on Processing so that I could add UI Elements to the tree display.

Monday, April 13th:
What follows is the beginning of a revised set of deadlines. I intend to complete the tasks listed here by Tuesday April 21st:

- Pre-parse Newick tree files to replace the given number-encoding with the appropriate scientific names, before sending the tree string to the regular PhyloWidget parser

- Retrieve images from Morphbank, based on searches by scientific name

- Incorporate a thumbnail version of the each image into the PhyloWidget tree display at its corresponding node, according to the diagram posted previously. (Later, allow multiple images per node and add UI interaction so that the user can scroll through the thumbnails for each.)
After this deadline, I plan to return to the current bug in my common name search to complete that portion of the code. I will also look at the way that PhyloWidget draws the trees with the alternative layouts, to determine how best to incorporate the images in those cases.

Saturday, April 18th:
I have successfully coded the portion of my project that involves pre-parsing Newick tree files so as to display the scientific names of the organisms rather than the index numbers.

Explanation:
Newick files exist in the following format, and their structure is defined by commas and parentheses:

```
#NEXUS
 BEGIN TREES;

TRANSLATE
 1 Alligator_mississippiensis,
 2 Chinchilla_brevicaudata,
 3 Felis_silvestris_catus,
 4 Balaenoptera_borealis,
 5 Oryctolagus_cuniculus,
 6 Balaenoptera_physalus,
 7 Mesocricetus_auratus,
 8 Meleagris_gallopavo,
 9 Lepisosteus_spatula,
 10 Camelus_dromedarius,
 11 Proechimys_guairae,
 12 Anas_platyrhynchos,
 13 Platichthys_flesus,
 14 Goosefish_lophinus,
 15 Hydrolagus_colliei,
 16 Canis_familiaris,
 17 Physeter_catodon,
 18 Acomys_cahirinus,
 19 Hystrix_cristata,
 20 Myocastor_coypus,
 21 Struthio_camelus,
 22 Myxine_glutinosa,
 23 Saimiri_sciureus,
 24 Cavia_porcellus,
 25 Elephas_maximus,
 26 Gadus_callarias,
 27 Cyprinus_carpio,
 28 Thunnus_thynnus,
```

```
 29 Equus_caballus,
 30 Crotalus_atrox,
 31 Batrachoididae,
 32 Myoxocephalus,
 33 Gallus_gallus,
 34 Capra_hircus,
 35 Oncorhynchus,
 36 Mus_musculus,
 37 Homo_sapiens,
 38 Anser_anser,
 39 Ovis_aires,
 40 Sus_scrofa,
 41 Bos_taurus,
 42 Mammalia,
 43 Macaca
 ;
 TREE tree_0 = [&R]
(((((((11,20),24),19),2,7,18,(36,23),37,43,5,(16,17,6,40),29,25,41,3,4,(34,39)),10),(33,8,21),30,1,15,(38,12
),9,((31,26),32,14,28,13,27,35)),22);
ENDBLOCK;
```

Currently, PhyloWidget only looks at the last line, where the structure of the tree is defined. As a result, the leaves are labeled with numbers rather than the scientific names. My pre-parser replaces all of the numbers in the last line with the corresponding scientific names, so that the true names become the label names stored in each leaf node. For the tree listed above, Figure 9 is now the new output.

Since PhyloWidget supports loading trees from files as well as from manual input, I've inserted the pre-parsing phase into both processes, of course with the expectation that both the file and the user-input be formatted according to the example above.

Week 13: April 19th – 25th

Monday, April 20th:
I am now able to search Morpbank.net for any given taxonomic name and return a LinkedList of all the images associated with that search. I have not yet integrated the code into PhyloWidget but intend to do so later today.

One of the reasons I have not done this is because many revisions were made to the code during March, which I am not interested in including in my version of the code. Since I download the code using an SVN client, I need to figure out how to download the code as it was before those revisions took place.

Once I am able to do that, the integration will consist of creating a variable to store these images within each node of the tree, determining when to call the search function with the node's taxonomic name, and displaying the resulting images within the current program.

Later:
I've integrated the image search into PhyloWidget. The images for each node are stored in the class PhyloNode as a LinkedList. Currently, when loading a tree, a search is conducted as each new node is created. This means that the loading process is quite slow, when really it should be possible to conduct the search after drawing the initial text-only display, or in a separate thread while the file continues to be parsed.

Next step is to display the images within the program as opposed to outside it in tiny little JFrame windows. That'll require some UI manipulation, which I was holding off on until the rest of the code was working. I'm also hoping to search some other databases, because Morphbank has very few images available for some categories of organisms -- at least that's true for the ones in the tree files I've been testing with.

Finally, one of the more amusing issues is the "Image Not Available" image. The Morphbank search only displays results with images, but unfortunately some of those images look like Figure 16. Obviously that isn't too useful. If Image objects are comparable, I could save one of these "Not Available" ones and test whether the returned image is equal to it before displaying it.

Wednesday, April 22nd:
I added an option in the menu toolbar for a Morphbank image search. The callback function retrieves the tree, gets a list of all of its leaves, and then passes each of their labels to the search function one by one.

In the meantime I've also attempted to sift through the massive network of code involved in rendering, so as to find out where exactly I jump in and write my code to insert the images before the label names of each leaf. No clear spot as of yet, but I'm knee deep in rendering code and at least starting to learn my way around.

Friday, April 24th:
Completed Morphbank image search and display.

Image URLs are now stored as NHX annotations on the node. Rather than downloading and storing each image in a LinkedList as before, the rendering program loads the URL at render time. This process runs in its own thread, so as to allow the user to continue interacting with the program while images are loaded.
Each node “outsources” the search process to an ImageSearcher2 object that coordinates searching and storing results.

Week 14: April 26th – May 2nd

Monday, April 27th:
In trying to expand my image search to include the nbii.gov image database, I realized I had a little more to learn about the process of posting/getting form data through Java.

Thursday, April 30th:
In the midst of preparing for my presentation, I tried to have as much of a finished product as possible, so I spent my time working on including NBII.gov in my image search. I output the string "txtSearch=term&Submit=Go" to
http://life.nbii.gov/search_results.php", with “term” being the given search taxon string.

When I get the response as an input stream, I search for the term "Total images", since it indicates that I'm about to reach the place in the html where the images are listed. I search for and save image URLs from then on, until I get to a line that contains "Xthumbnail", since that indicates that I've gone past the section with the images I want.

I successfully integrated NBII search into PhyloWidget in time for my presentation.

Week 15: May 3rd – 9th

Monday, May 4th:
UBIO Common Name Search is complete!

I've spent the past 2 days trying to work around strange bugs and get a working common name search, and I got it. By Sunday I had the code parsing multiple pages in search of the right information and printing out the resulting common name in the console. Today I changed some of the set and get functions relating to labels so that a user could toggle whether the common or scientific name is displayed, and I now have the names actually displaying as labels on the rendered tree. I even figured out how to display a little message "Done searching for images!" just like there's a "Finished Loading Image!" message.

See Figure 14 for a sample image from after the name search/display has been done. It can't find every name, but it gets most of them.

I also spent some time reorganizing the toolbar menus so that they feel a little more intuitive and take fewer clicks to get to what you want. I felt the original structure of the toolbar lead to a lot of overlap and confusion about where a desired option was located. I organized my menus as follows:

Tree
- Layout (rectangular, circular, diagonal, unrooted)
- Node (images and other annotations)
- Label (unique, clade, show all, common name search)
- Performance (anti-aliasing, animation)
- Zoom

Format
- Font

Style (size, scaling, angles)
Structure (sorting, flipping, branch lengths, etc.)

I think now the functionality of the program is actually arranged in a way that makes sense based on what the different options control and how they change the display of the tree. I also spaced them out a bit more, since I found that the original toolbar options were crowded into the left corner.

Friday, May 8th:
I thought through a couple different ways to allow users to see multiple images for a given node, and I decided that the simplest user interaction was to have the image change to the next one when clicked on. It wasn't actually as simple to implement as it is to use.

I created a Rectangle class extending the UIObject class, which the EventManager class sends events to. The rectangle also has a reference to the overall PApplet (Processing Applet), so as to control drawing this invisible rectangle on the screen. On a mouse click event, the EventManager sends the event to whatever UIObjects it knows of, and waits for one of them to handle it before resorting to other options. The Rectangle class handles MOUSE_RELEASED events that fall inside the area of the image.

First roadblock - either the rectangles were responding to clicks indiscriminately, no matter where they were, or they weren't responding at all. Then when debugging through I realized the EventManager had references to a lot more UIObjects than seemed reasonable... until I realized that for every node on every new frame I was creating a new rectangle instead of updating that node's existing one. That led to thousands of rectangles in a very short time and caused much of the strange click-response I'd been getting.

Finding the click area itself was a bit of a task, because there aren't clear variables defining it. The function drawing the image is called using local coordinates, but my rectangle needed to be called using world coordinates. This is where my CIS 460 knowledge came in handy, because otherwise I wouldn't have even known the difference let alone how to handle it. I was able to find variables called realX and realY in world coordinates, which I thought would be the solution to my problem, except for some reason only the bottom left side of the image was responding to clicks. A lot of debugging and detective work lead me to find out that that the realX and realY coordinates are actually that location of the center of the node dot, not the top left corner of the image as I had previously thought. So even though my rectangle was drawn in the right place (tested in color, so I'd know where it was), the code deciding which clicks to recognize was handling the wrong area.

Once I figured that out, the next step was to try and fix it. The image is drawn just to the

right of that node dot, but how far? Some quick math got me a rough estimate, except that value changes depending on the zoom level. What solved it for me was the variable dotWidth, which changed along with the zoom. Turns out the image's top left side is approximately one "dot-width" away from the realX and realY coordinates. Since the dot hits at the center of the image, vertically, I subtracted half the height from the y coordinate and finally ended up with the correct "click area."

The last step was to handle actually making a click do something other than print out "click!" which is what I'd been using to test it until then. After some quick searching around, I found that the ImageSearcher2 object has a next() function. So I called that. Nothing happened. Then I added a line for loadImageURL() and finally I had changing images on clicks! Except when it got to the end of the available images it stopped. So I added a check to restart from the first when it got to the end, and that is the long but exciting story about how I got my last feature working properly.

“Tree of Life” Teaching Tool

Appendix: Images

Figure 1

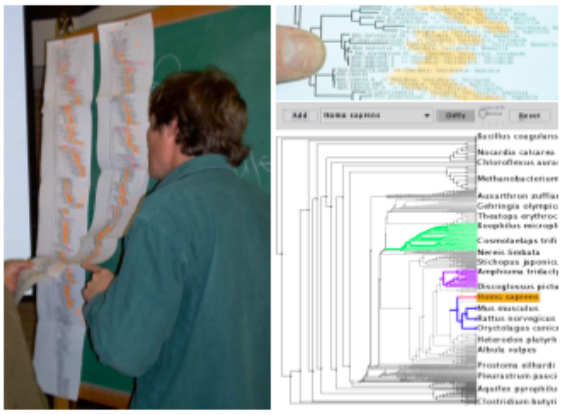
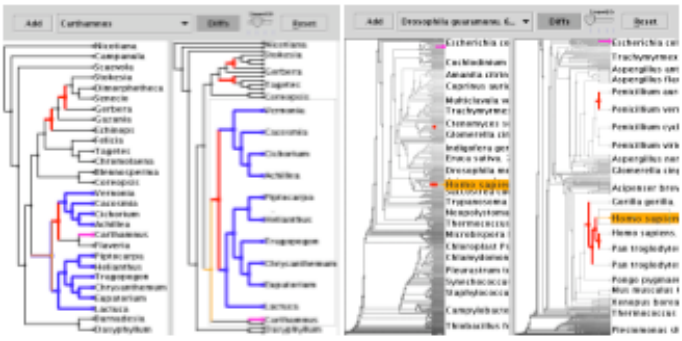


Figure 1: Left, Right Top: Biologists faced with inadequate tools for comparing large trees have fallen back on paper, tape, and highlighter pens. Right Bottom: TreeJuxtaposer is a scalable tool for interactive exploration and comparison of trees.

Figure 2



Tree comparison between two variants of a single phylogenetic reconstruction run, with the exact location of structural differences (in biological terms, nonmonophyletic clades) marked in red. The left tree is in the undistorted overview position, while parts of the right side have been expanded. **Left:** A small 55-node tree. **Right:** A larger 1600 node tree.

Figure 3

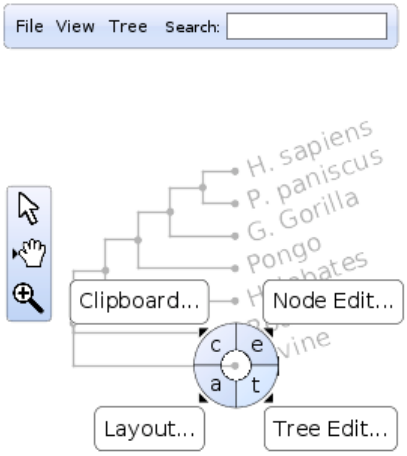


Figure 4

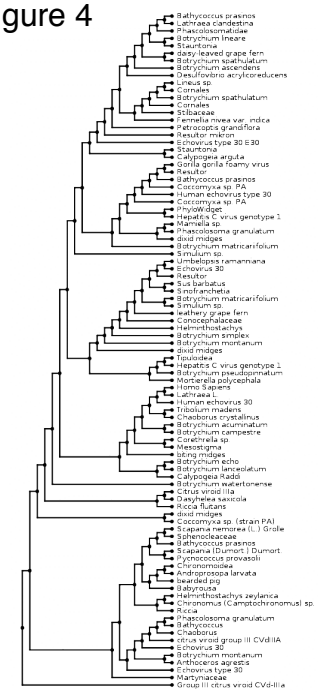


Figure 5

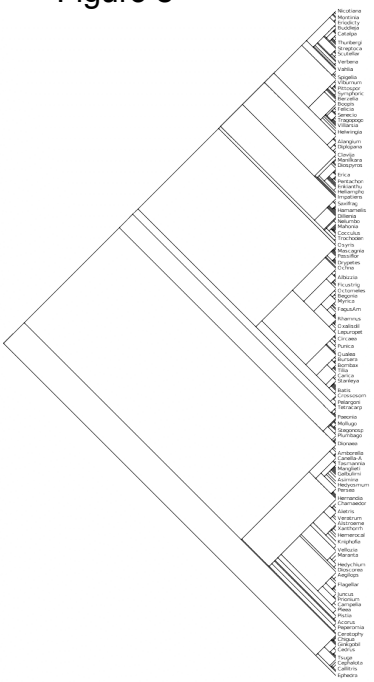


Figure 6

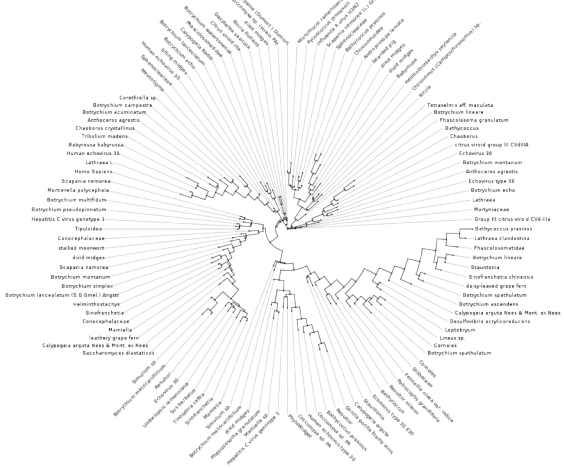


Figure 7

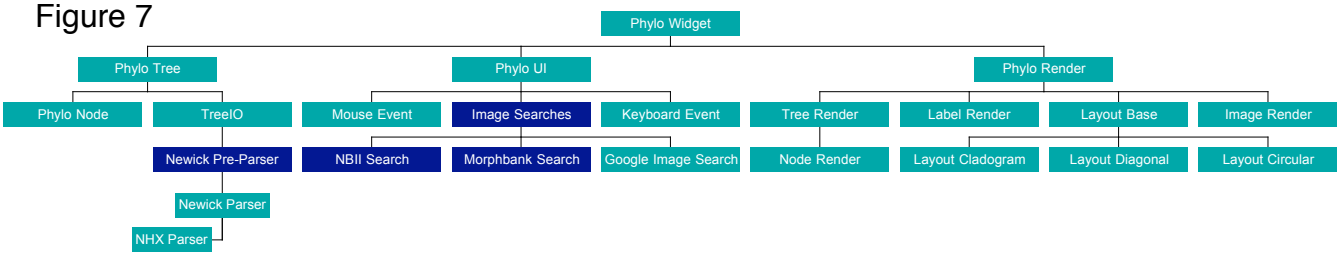


Figure 8

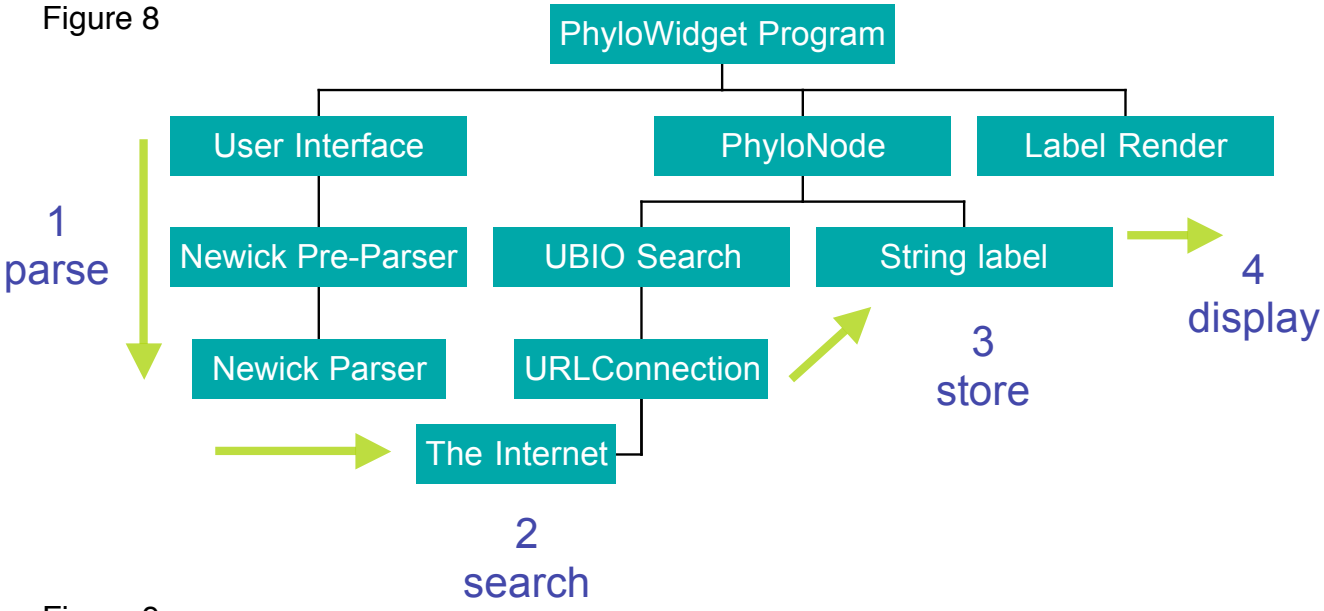


Figure 9

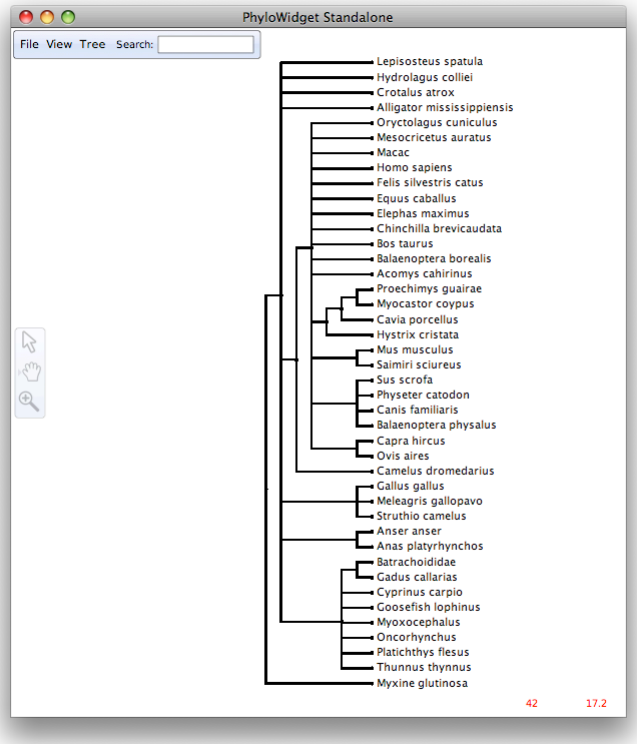


Figure 10

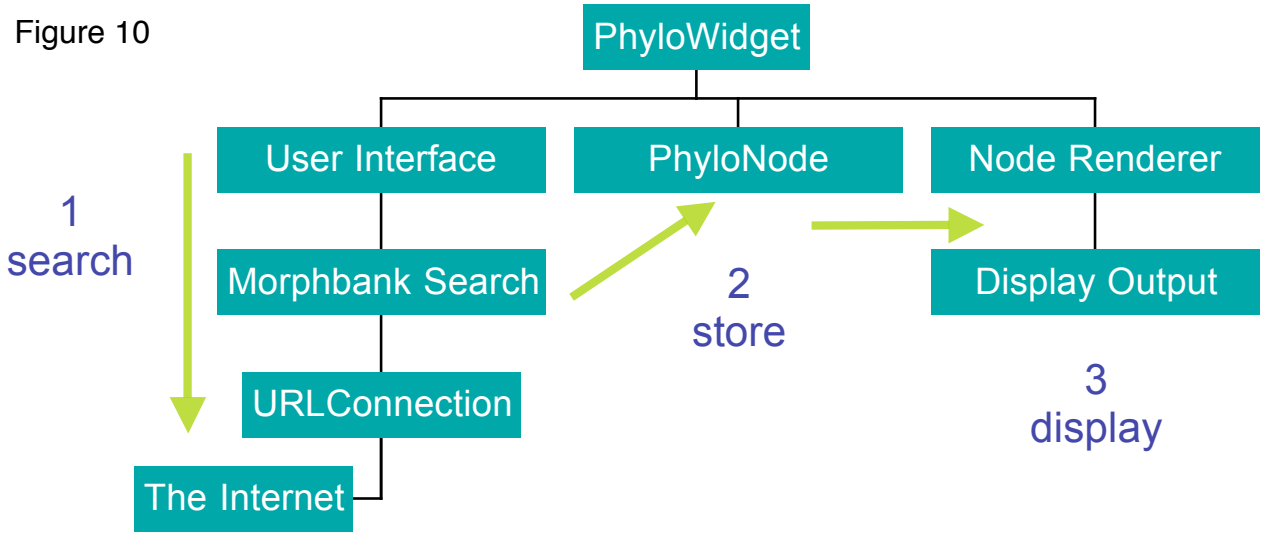


Figure 11

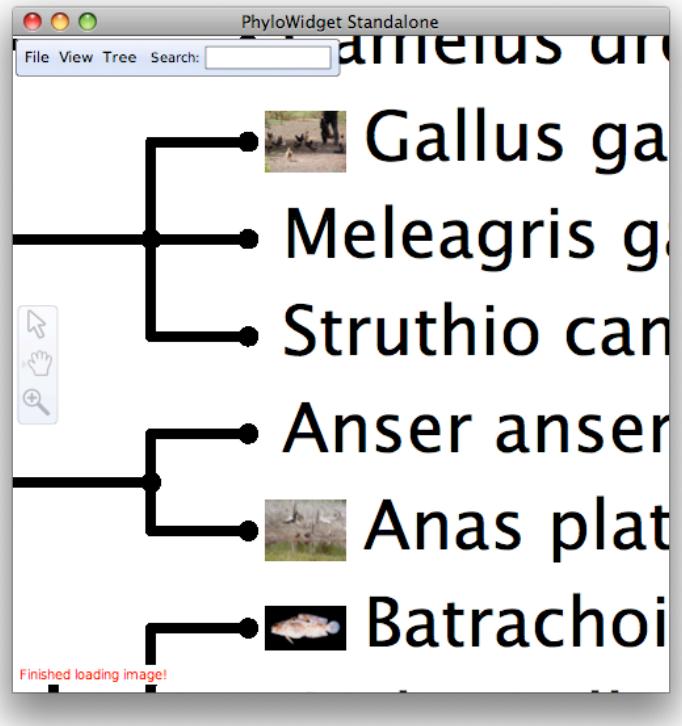


Figure 12: a, b, and c

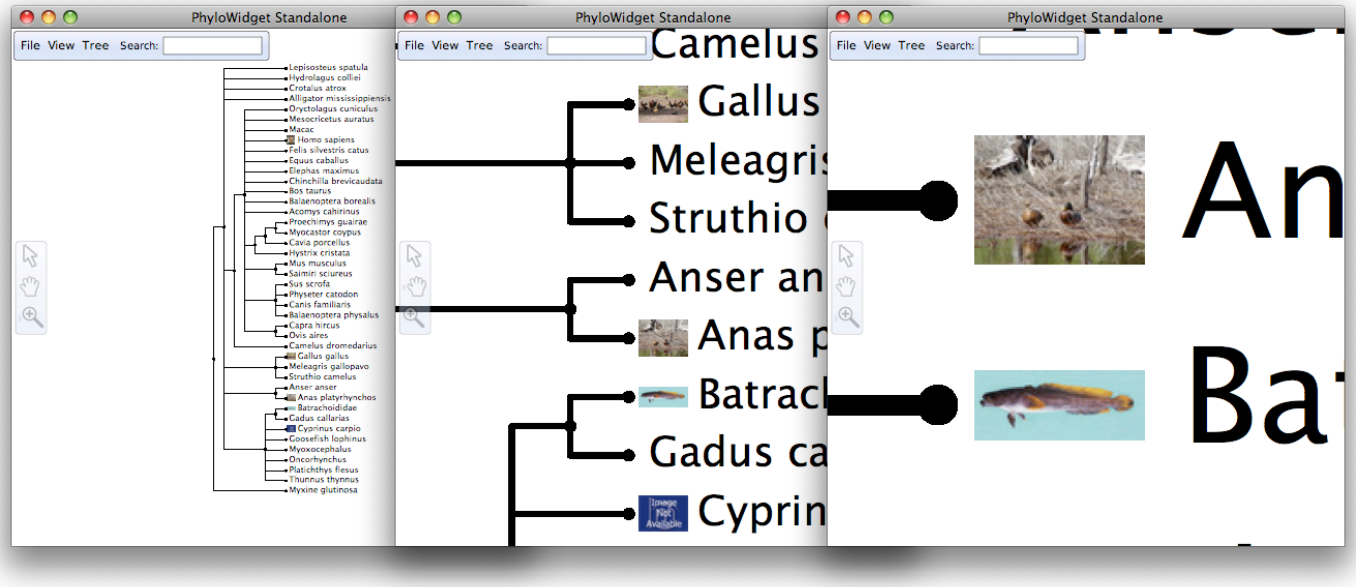


Figure 13

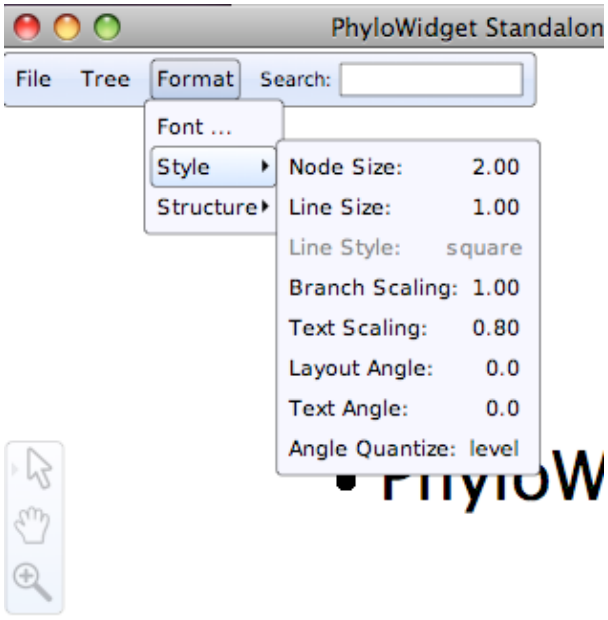


Figure 14

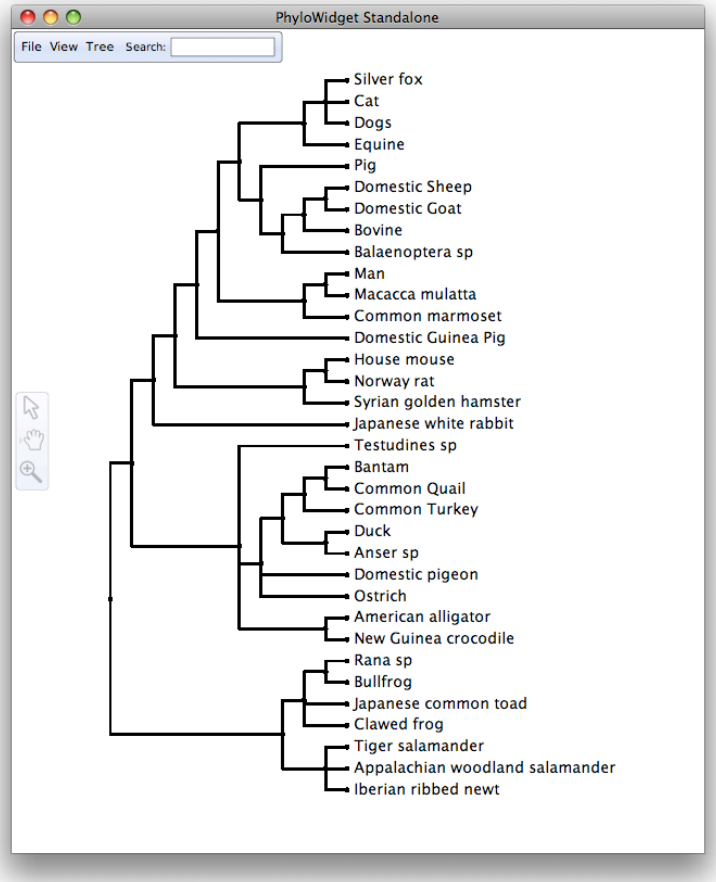


Figure 15

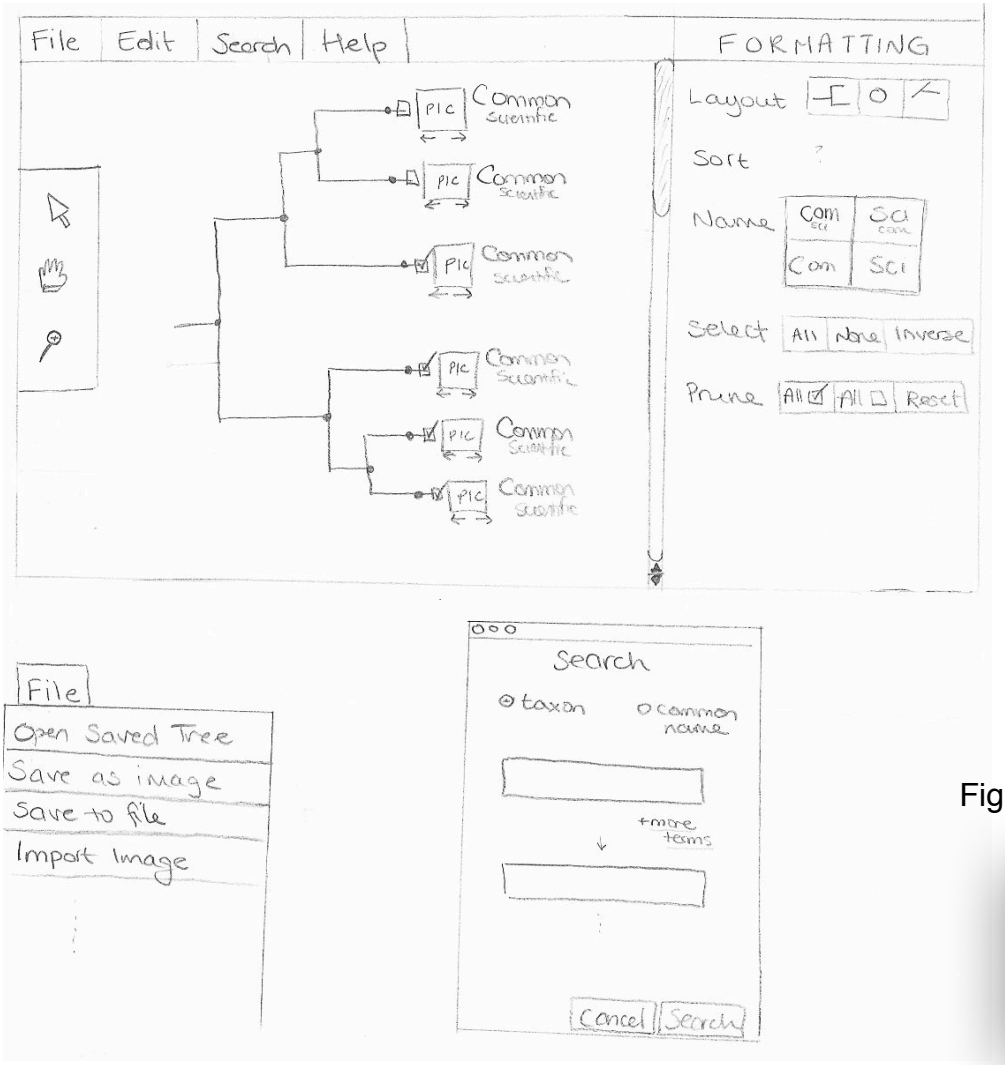


Figure 16