Research: yuml2graphviz (Summer 2015, 1 credit) Tom Taylor 07/31/2015

The two major things that I learned related to this research project were the complexity of data parsing and in making sure that the requirements are fully understood before starting to code. I now understand why the use of preexisting parsers is so prevalent, as I learned it can be very difficult to fully and properly parse, even for a relatively simple language like yUML, from scratch. I was told about, shown, and looked into one such parser known as ANTLR. Unfortunately, with time constraints and ANTLR's complexity I was unable to pursue using that approach to parse the yUML input. My first attempt at coding this was from working with yUML class diagram samples directly. In this scenario both the classes and the relationship between them seemed to be an integrated unit. This lead to the idea of a left hand side (lhs) and right hand side (rhs) class as the relationship is directional. This was the approach taken with the first coding attempt. After more discussions I learned that all of the classes could be input separately first, this removed the concept of lhs or rhs as it related to classes, and then the relationships entered separately with just a lhs and rhs reference to the appropriate preexisting classes. This was the approach taken with the second coding attempt. The upside to this was that I ended up with simpler code in the long run because I could then use the same code for any class. Also, I was then actually able to make the code handle either input scenario. I have run into this having to recode a majority of a first attempt before. It seems to always produce a better and cleaner final solution. It just seems difficult to get all of the complexities of a solution truly worked out ahead of time, no matter how hard you try, as you run into things you didn't consider or fully understand once you start writing the program.

As far as the code I have provided goes, the parsing definitely needs to be implemented in a much more thorough and robust manner. That would include for both the class parsing and the relationship parsing. Along with that, there was not much implementation done when dealing with color and notes. The color tags are being stripped and are stored in the appropriate class structure. All of the notes have been pushed into a separate container, to be dealt with separately, while the input string was parsed. However, neither color or notes are actually making it into the generated dot file. Perhaps most importantly, the testing done against the implemented code was not anywhere near fully complete and all encompassing.

However, a lot was learned along the way and should go a long ways in helping with the production solution. The structure of the dot file currently does a pretty good job at this point of emulating a yUML generated diagram. Some of the keys to this are at the beginning of the generated output file:

digraph G {

nodesep=0.75

ranksep=1.5

node[shape=record, style=filled, fillcolor=gray95, fontname=Verdana, fontsize=14]

edge[dir=both,arrowhead=none, arrowtail=none, fontname=Verdana, fontsize=12, labeldistance=2.0]

The nodesep is the spacing between the classes horizontally. The ranksep is the spacing between the nodes vertically. This one is very important as otherwise the edges can be too short which makes the classes too close together. This is a problem both visually and because the edge labels (cardinality, etc.) look better with extra space as you manipulate them away from "arrowheads". The labeldistance parameter is what does this manipulation. It is a scalar that moves the label away from the edge contact point. This is important as you need the labels to be outside of the "arrowhead" space for readability. There is also a labelangle parameter, but that one really needs to be changed somewhat on an edge by edge basis as it works off of the starting angle of the edge. The above settings do a pretty good approximation of what is currently being done at yUML and the diagram looks pretty good. The only trouble was with font size, I really wanted to go with 12 and 10 point but for some reason they look just awful at those particular point sizes for Verdana. Another important improvement was making the nodes of shape record as it really was the key to getting the nodes to look like a proper class as expected on a class diagram.

Next, the concept of using the class name as the node label was utilized and these labels are then used when creating the relationships between the two classes:

Category[label = "{Category}"]

Product[label = "{Product}"]

Category->Product[arrowtail=odiamond, arrowhead=vee]

However, this is a little deceiving as the label is actually left of the bracket and the keyword label means what text is actually going inside of the node (class name, attributes, methods).

Another area that should help is looking through the two parsing functions related to parsing the relationships. This includes line type, arrowhead type, and node labels. I included comments related to what is being done and why and this was gleaned from research and trial and error. I could not find any definitive documentation on how to properly build the yUML relationships. Specifically, I could find no list of valid symbols and what they mean or anything about how to properly order things inside the relationship itself. Early on we had discussed and found yUML grammar online. However, at this point I do not think that is of much use as it does not include or talk about any of the actual characters, and their meaning, used to create yUML at all.

I also think that spending some time going through this code in general will be helpful. The two classes that are the class and relationship structures should be a pretty good starting point for what is needed. Even though the parsing in general is not being done at all like it should be done in the final solution , all of the parsing code is commented and should help with what symbols need to be looked for, why, and in what order. The same goes for the code that generates the dot file, it methodically builds that file so there is usefulness there as well. It handles building the wide range of nodes, from nodes just containing class name to ones containing the name along with multiple attributes and methods. Here is a link to the most useful Graphviz document I found while working on this: http://www.graphviz.org/doc/info/attrs.html#h:undir\_note