Connor Jackson, Adam Claxton

Assignment #3 CSE2102

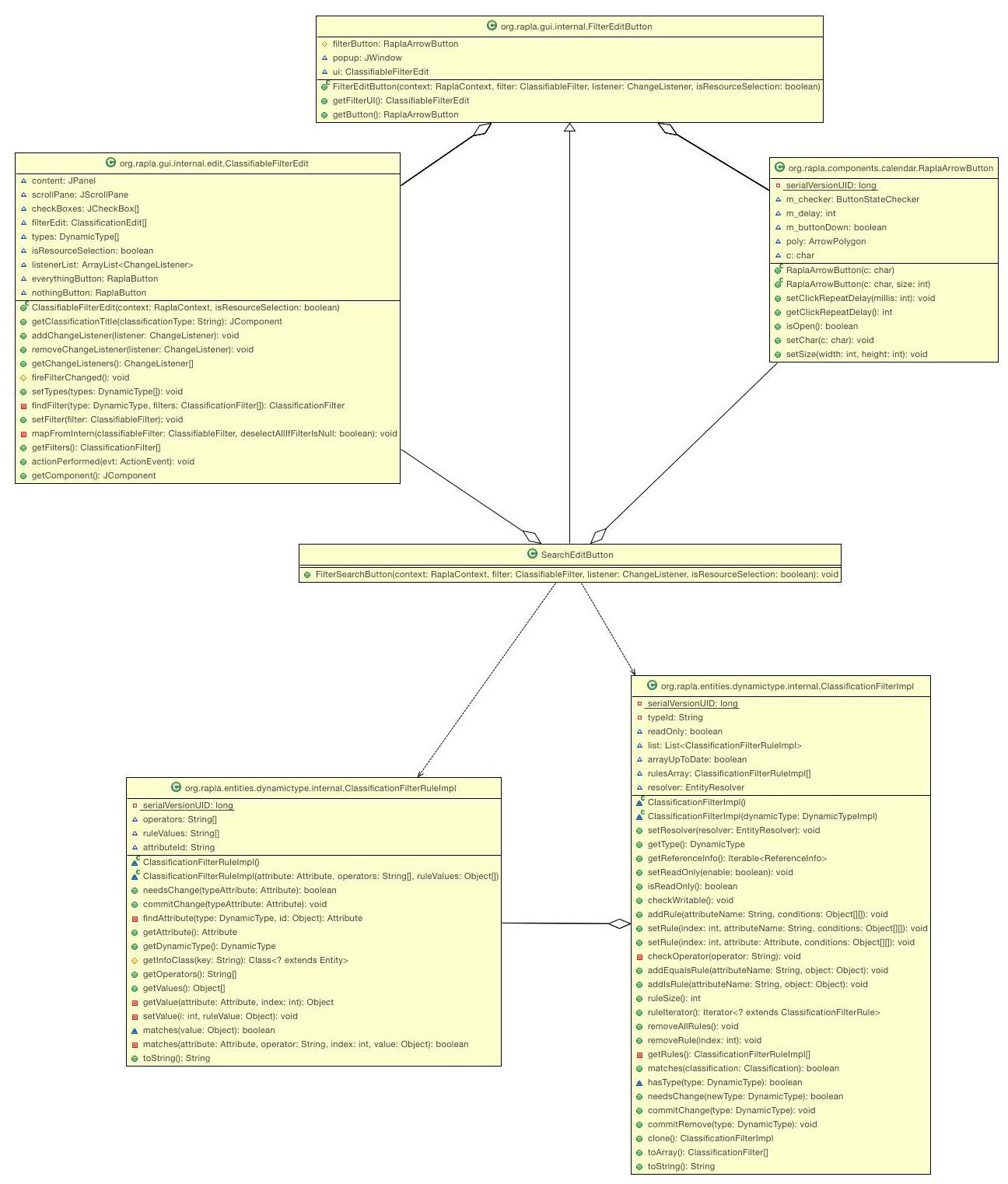
Rapla Scheduler – Fast Resource Search

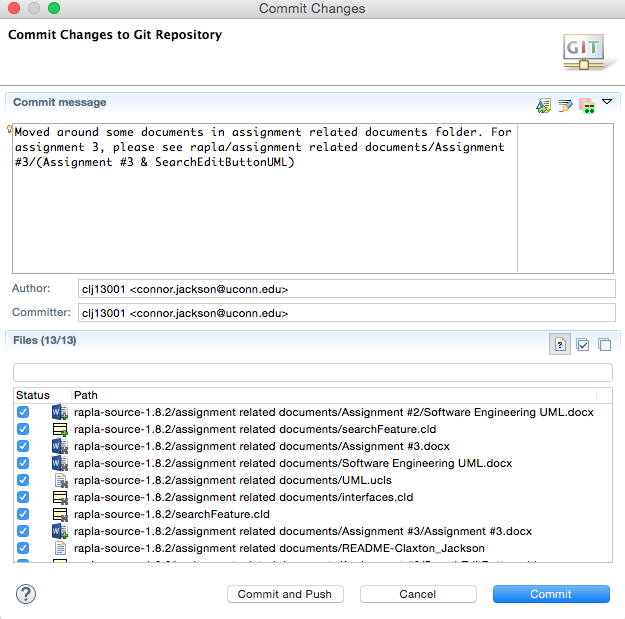
October 11, 2015

1. In this project, we will be adding a fast search feature to the resources section of this application. We are interpreting this as picking a type of resource, choosing an attribute to search by, and typing in what we want to search. Then, in the calendar view, only events containing that resource-attribute match will be displayed. This is very similar to the filter feature, except you have to uncheck every single resource you don’t want to see if you only want to view one. Our addition will make displaying a single resource much easier.  
     
   Since we can use a similar implementation as the filter feature, we could use inheritance from FilterEditButton located in org.rapla.gui.internal. This creates a small button from the RaplaArrowButton class that looks like such:



We would have to change the text to read “search” though.   
  
Along with this, we would like to use the class ClassifibleFilterEdit located in org.rapla.gui.internal.edit. This class seems to help create text fields and combo boxes inside the popup a filter button would have, which helps in our search feature. We would like our user to choose a resource to search through a combo box, then an attribute through a combo box, then type in their criteria through a text field.  
  
We are also thinking about using ClassificationFilterRuleImpl & ClassificationFilterImpl from org.rapla.dynamictype.internal or ClassificationFilterRule & ClassificationFilter from org.rapla.dynamictype. These parts of the program seem to contain the code that uses filter rules that choose which events in the Rapla interface fit defined rules. In this case, we can manipulate it to match events with search criteria.  
  
Lastly, we will need to change the code a bit in the section of this project that actually adds each part of the program to the interface frame. This is simply where we will add the search arrow button.

1. The existing code helped a lot in our design process. Our first gut reaction to working on this assignment at first was to explore the program and see what features already existed, from a user perspective. Once we found how closely resembling the filter feature was to search, we knew we should change that around to make our work progress well. This way, we won’t have to develop any algorithms for searching and just use ones that already work perfectly fit with the program.  
     
   Also, the project was designed in such a way that most of the code is generalized and can be reused in many different cases. This makes it very easy for extra add-ons to be implemented, as it seems designed to be easy for others to collaborate on.
2. At this point in our design, we would like to design a class that uses bits from each of the classes mentioned in 1. Class diagrams are usually designed so that the parent class of a diagram is located at the top. Since our search button will branch from the filter button, the filter button will be located above the search button.  
     
   Then, ClassifiableFilterEdit and RaplaArrowButton are in between the filter button and search button. That is because both buttons aggregate with these classes, and this placement prevents any arrows from hitting each other.  
     
   Then there is the search button and the two Impl classes that we would like to use as a part of our code. Since we want to call on the filter functions included in these Impl classes, we use a dependency arrow to show that our search button depends on these two classes. Since this relation is independent from the filter button, we put the Impl classes at the very bottom to show that their significance only exists with our search button.
3. First, our diagram shows that SearchEditButton extends FilterEditButton. We want to inherit all methods within the FilterEditButton because SearchEditButton will work the same way; has an arrow, we click on it, a popup shows, and you choose/type search criteria. Some specialization will have to take place, so we will probably end up editing the constructor method a bit in SearchEditButton.  
     
   Next, we use aggregation to show that SearchEditButton has ClassifiableFilterEdit and RaplaArrowButton. It contains an instance of RaplaArrowButton, because the button has a literal arrow printed on it and we will use this button to open/close the search popup. It has an instance of ClassifiableFilterEdit because this class contains TextFields and JComboBoxes; these will be used to enter user input and choose search attributes (respectively).  
     
   Finally, we use dependency to show that SearchEditButton depends on ClassificationFilterRuleImpl and ClassificationFilterImpl. These classes contain code related to filtering resource types; it checks events in the current view for a filtered resource type and only shows events that meet this criterion. Since search works basically the same way, we can use this algorithm process to implement our feature. We only need to call on these methods, we use dependency.
4.   
   Here’s our UML diagram: The .cld and .jpg file are both separately included in our repository, because this is probably difficult to view.



And here is the screenshot of us committing our files. See the message to find their location.