Connor Jackson, Adam Claxton

Assignment #3 CSE2102

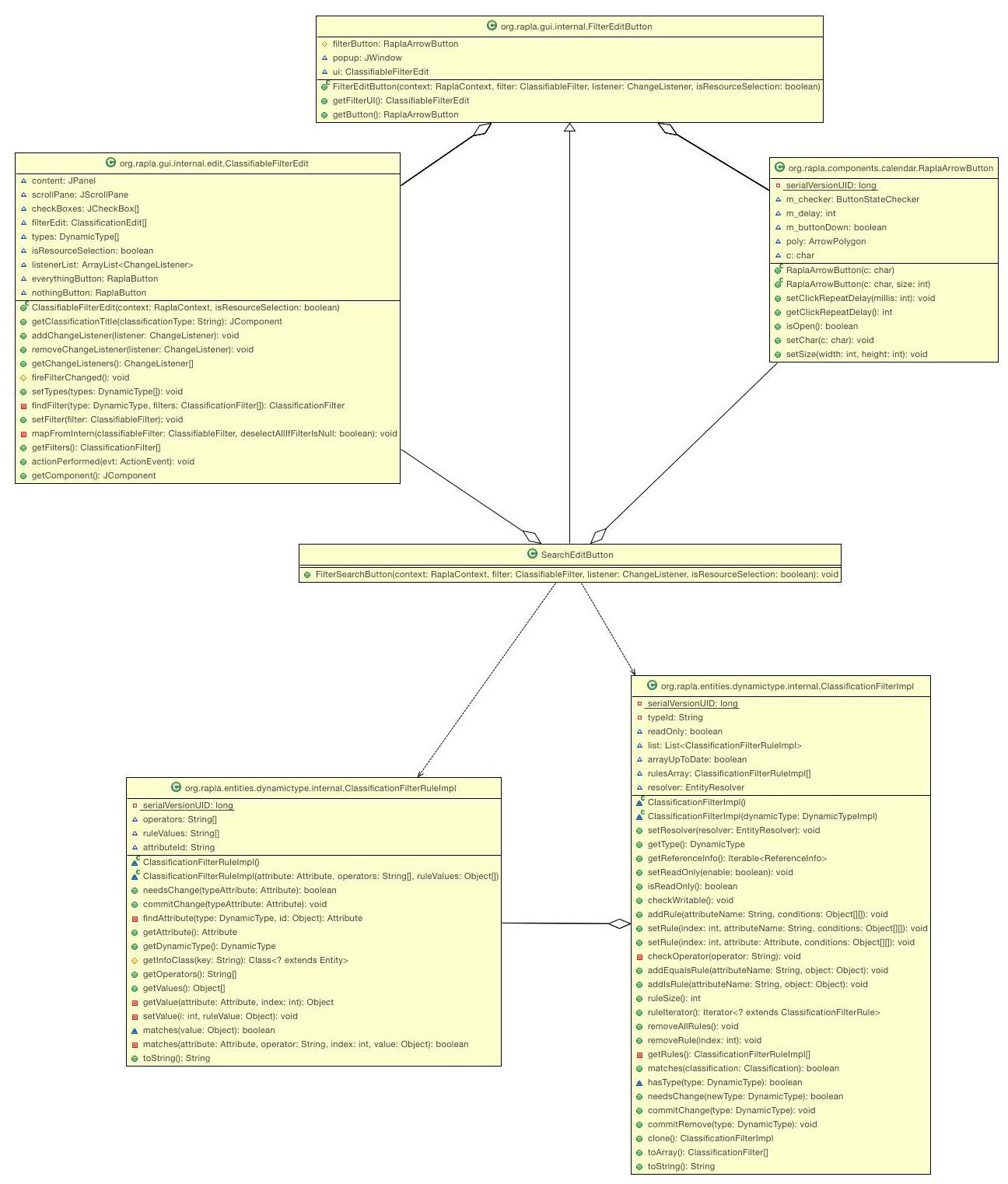
Rapla Scheduler – Fast Resource Search

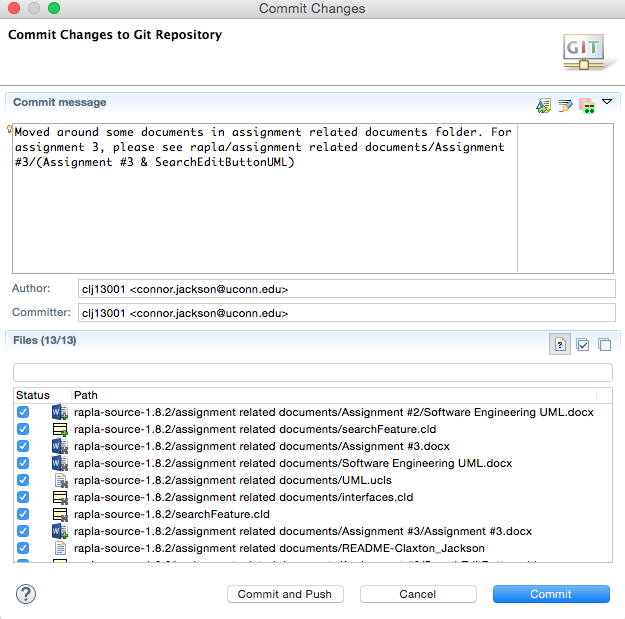
October 11, 2015

1. In this project, we will be adding a fast search feature to the resources section of Rapla. We are interpreting this as: picking a type of resource, choosing an attribute to search by, and typing in search criterion. Then, in the calendar view, only events containing that resource-attribute match will be displayed. This is very similar to the filter feature, except filter requires unchecking every single resource you don’t want to see (if you only want to view one kind of resource). Our addition will make filtering/searching for a single resource much easier. We will refer to it as SearchEditButton.  
     
   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 TextFields and JComboBoxes inside the popup a filter button would have, which would also be needed in our SearchEditButton. We would like our user to choose a resource to search by using a JComboBox, then an attribute through another JComboBox, then type in their criteria through a TextField.  
  
We are also thinking about using ClassificationFilterRuleImpl & ClassificationFilterImpl from org.rapla.dynamictype.internal. These parts of the program seem to contain the code that uses filter rules that choose which events in the Rapla interface are displayed. In this case, we can manipulate it to work with SearchEditButton also.  
  
Lastly, we will need to change the code a bit in the section of this project that actually adds buttons to the interface frame. This is simply where we will add SearchEditButton to a certain spot in the GUI.

1. The existing code helped a lot in our design process. Our first gut reaction to working on this assignment was to explore Rapla and see what features already existed, from a user perspective. Once we found how closely resembling the filter feature was to a search feature, we knew we should specialize it to make our addition. This way, we won’t have to develop any new algorithms for searching and just use ones that already fit 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. For example, RaplaArrowButton simply allows the user to create a box with any text, but the box contains an arrow that suggests a drop-down box to appear when clicked. This class can be extended to others to create many different types of RaplaArrowButtons.
2. At this point in our design, we would like to design our UML to map out in a formal way how SearchEditButton will relate to all the other classes previously mentioned. Class diagrams are usually designed so that the parent at the highest level of hierarchy is located at the top of the diagram, and its subsequent children are immediately below (and so on and so forth). Since SearchEditButton will branch from the FilterEditButton, FilterEditButton will be located above SearchEditButton.  
     
   Then, ClassifiableFilterEdit and RaplaArrowButton are in between FilterEditButton and SearchEditButton. That is because both buttons aggregate with these classes, and this placement prevents any arrows from intersecting.  
     
   Then there are ClassificationFilterRuleImpl and ClassificationFilterImpl. We want to call on functions included in these classes, so we use a dependency arrow to show that SearchEditButton depends on these two classes. Since this relation is independent from FilterEditButton, we put these classes at the very bottom to show that their significance only exists with SearchEditButton.
3. First, our diagram shows that SearchEditButton extends FilterEditButton. We want to inherit all methods and private variables within 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 RaplaArrowButton and ClassifiableFilterEdit. 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 SearchEditButton. We only need to call on these methods, so 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 in this file.



And here is the screenshot of us committing our files. We will organize all our files to be located in a folder titles “assignment related documents” so all our specific contributions stay independent from files the Rapla developers created.