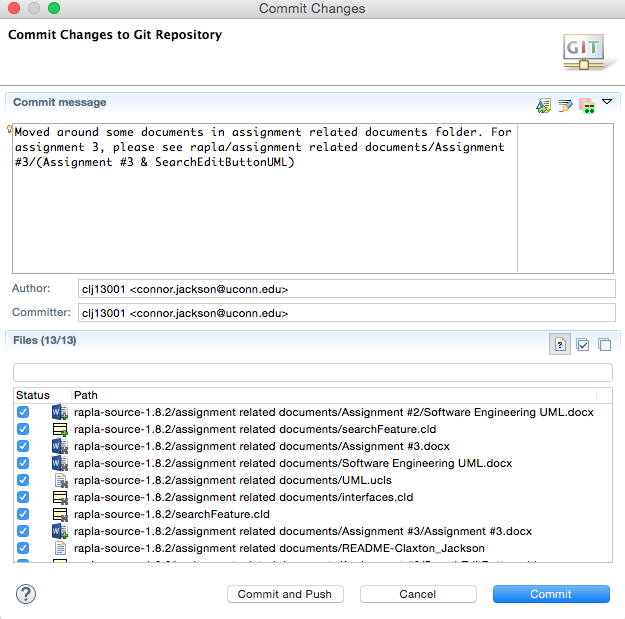
Connor Jackson, Adam Claxton

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: creating a search bar where a user types in search criteria, and the input it compared with all resources that exist. Then, in the calendar view, only events containing those resources will be displayed. This is very similar to the filter feature, except filter requires choosing resource type and attribute, which you have to define manually for every resource. Our addition will make searching for multiple resource types containing the same keyword much more efficient. We will refer to it as SearchTextField.  
     
   Rapla contains a class called TextField found in org.rapla.gui.internal.edit.fields. This is a specialization of a JTextField, and will allow the user to type in search criteria. We can extend this class and take all the properties that the class is designed to handle, and specialize it so it works with search algorithms.  
     
   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. Since this users hide/show methods for events, especially in junction with filtering, we can use this to SearchTextField’s advantage.  
     
   Lastly, we will need to change the code a bit in ResourceSelection from org.rapla.gui.internal. This is simply where we will add SearchTextField above the resource section.
2. 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 could specialize it to make our search 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, TextField simply allows the user to create a JTextField, and it implements rapla-specific interfaces that are referred to in numerous other classes. This allows it to be seamlessly integrated without making sweeping changes to many different classes.
3. At this point in our design, we would like to design our UML to map out in a formal way how SearchTextField 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 SearchTextField will branch from the TextField, TextField will be located above SearchTextField.  
     
   Also at the same level of the UML diagram (above SearchTextField) is ResourceSelection. This class will contain an instance of SearchTextField as a member variable, so it seems to make sense to place it above SearchTextField since SearchTextField is “contained” within ResourceSelection.  
     
   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 SearchTextField depends on these two classes. Since this relation is independent from TextField, we put these classes at the very bottom to show that their significance only exists with SearchTextField.
4. First, our diagram shows that SearchTextField extends TextField. We want to inherit all methods and private variables within TextField because SearchTextField will work the same way; it will allow for user input. Some specialization will have to take place, to implement the search, so we will probably end up editing the constructor method a bit in SearchTextField.  
     
   Next, we use aggregation to show that ResourceSelection has SearchTextField. ResourceSelection is the class where SearchTextField will be added to the frame, so that the user can actually see the field and click in it, type, etc. The class has an instance of FilterEditButton as a member variable, and later adds it to the GUI; similarly, we will add an instance of SearchTextField to the member variables so it too can be added to the GUI.  
     
   Finally, we use dependency to show that SearchTextField 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 SearchTextField. We only need to call on these methods, so we use dependency.
5.   
   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.