Design Discussion

MVC Architecture with Active Model

Architecturally the system is based on the MVC pattern with an active model. This pattern seems like a good fit for the system because it can be broken neatly into the layers described by the pattern and the model needs to update regularly, prompting the user interface to change.

Data is stored in the Location class and updated through WeatherService. Both of these classes have been placed in the model package because they only hold or update system state and do not depend on other modules. WeatherService is observable, so other parts of the system can be notified when new data is available. Setting up the classes in this way eliminates dependency between the model and the rest of the system, which means that the model never has to be changed in response to changes elsewhere in the system.

The controller package should respond to user input and instruct the model or view to update as needed[1]. The controller and listener classes between them satisfy this requirement. The Controller class is a general bag of functionality for routing messages through the system, and the Listener and WeatherServiceComboBoxListener classes respond to to user input from the view.

Finally, the view package holds classes which update different parts of the view. For example, the EastPanel class defines the right hand side of the view, and contains a number of buttons and dropdowns. If new data arrives at the WeatherService class this layer can respond to it and Monitors can update themselves as necessary.

Factory Pattern

The factory pattern is used to create new monitors which have the same construction inputs. By using the design suggested by the site OODesign[2] monitors can be added to the factory just by subclassing the Monitor class, then asking the Factory to register them for use. A Factory was used because it seems like an appropriate way of creating similar objects cleanly, and the method described in the article makes it amazingly easy to add a new monitor for creation.
The factory could have been made abstract but it seems unlikely that a whole new family of monitors could be designed so differently that the stock monitor class was unusable.

[1]The MVC Architectural Pattern Page 17, lecture notes
[2] http://www.oodesign.com/factory-pattern.html

Adaptor Pattern

An example of the Adaptor pattern in the system is the WeatherService class, which converts the calls in the *Stub classes into a consistent interface. This allows new services to be added as required, so long as they have a wrapping class which implements the interface.

Future Improvements

Controller Cohesion

Based on the description of class cohesion in lectures[3], the controller and listener classes seem to have low cohesion. To increase cohesiveness the listener could be split into many different listeners (similar to the level of WeatherServiceComboBoxListener) where each listener deals with a different button or component and only contain methods to deal with those events. Splitting the controller in this way would make it easier to find errors in listeners and make understanding the system much easier.

Abstract vs Concrete Factory

The monitor factory could be made abstract so that the controller could start relying on a different type of monitor if needed.

[3] Principles of OO and Design, Page 18