# Design decisions

1. Use a MVC architecture, since the application is inherently interactive, contains a model, a controlling part and several views.   
   With this architecture we want to decouple views from the model and the controller logic.  
   This will help us to achieve flexibility and reuse. More specific, if we take a new controller, we do not have to modify the model or view part. Also, when using a new GUI library, we can replace the view package and reuse the controller and model package. For this last reuse example, there is one problem left: the model elements inherit from the DefaultMutableTreeNode class. This is done because this DefaultMutableTreeNode class already contains tree-functionality and was probably chosen by Luc and Vladimir for this purpose. The ‘bad’ thing about this is that now the model classes are mixed with some Java GUI class, while it should be separated.
2. Make class ModelCollection a singleton, so that both the View classes as well as the Controller classes can directly access the model. With this approach it is not needed to pass a reference to the single instance of the model collection to all view and controller classes.
3. Make ModelCollection a Subject that is being observed by the Browser and PanelDiagramPane class. The reason for this is that Browser and PanelDiagramPane are only two examples of views. Later on, one might decide to introduce another view (e.g., a Pan & Zoom view). To make sure that this new view also contains an up to date version of the model, we use the Observer pattern to update the views.  
   Changes made to the model, via the ModelController (explained below), will call notifyObservers on the ModelCollection (the Subject), so that he can call update on the Observers.
4. Introduction of ModelController. This class contains the maintaining logic for all model elements. It gives a single point of access to change the model, and a single point of responsibility to call notifyObservers (as explained above).