Design Rationale

In our class diagram design, we have attempted to apply the passive Model-View-Controller (MVC) pattern. As the frontend needs to continuously update the display of the data that is generated by the backend, the UI and data processing components need to somehow know about one another. However, if the components directly depend on each other, we will have to test for both parts even though we may only want to test one or the other. This situation violates the Acyclic Dependency Principle (ADP) because it creates a cycle in the dependency structure (Martin, 1997). From an example illustrated by Martin (1997), they suggest breaking the cycle by creating a package that both components will depend on. This is done by moving the class(es) that they both depend upon into that new package. Hence, the controller aspect of the MVC pattern is introduced so that it can inform the UI to update the display. In this case, we have a controller package with the following classes: AppController (starts up the view), PatientNameController (prints the patients' names), AddPatientController, RemovePatientController and ViewPatientController. We are aware of the complexity that this pattern may introduce, thus as a result our code, in particular the Controller package may be difficult to manage during functionality extension. Despite this however, the pattern allows our data processing components (Model) to be tested independently of the presentation. (As mentioned in the week 8 lecture notes)

At the class level, we have used the observer pattern as well in the Model package (task2.application). Martin (2000) mentions that often one element of a design (actor) has to take some form of action when another element in the design (detector) sees that an event has happened, but we do not want the detector to know about the actor. In this case the actor, also known as the subject is the ObservationProcessor class. The detector, also known as the observer, is the PatientInfo class. When it is time to query the server, the update method in ObservationProcessor will be called to retrieve the data and update the PatientInfo class.

References

Martin, C.R. (1997, December). *Granularity*. Retrieved from https://drive.google.com/file/d/0BwhCYaYDn8EgOGM2ZGFhNmYtNmE4ZS00OGY5LWFkZTYtMjE0ZGNjODQ0MjEx/view

Martin, C.R. (2000). *Design Principles and Design Patterns*. Retrieved from https://drive.google.com/file/d/0BwhCYaYDn8EgODUxZTJhOWEtMTZlMi00OWRiLTg0ZmEtZWQ5ODRlY2RmNDlk/view

The Model-View-Controller Architectural Pattern [Lecture notes]. Retrieved from https://lms.monash.edu/pluginfile.php/10698523/mod_resource/content/1/MVC.pdf