Team MIVC

|  |  |
| --- | --- |
| Project: | MIVC |
| Due: | 9-30-2013 |
| Members: | Piper Chester |
|  | Geoff Berl |
|  | Colin Ferris |
|  | Ty McGinnis-Kennedy |
|  | Allen Thomas |

*Insert table of contents here after document is complete…*

Narrative …how it all works

As software engineers we often face the daunting task of building an application that is functional, extensible, closed for alterations and meets the customer requirements. This should all be done before any code construction is performed. We attempted to design the most simplistic way possible that was still able to meet the requirements of the customer and allow for later extensibility. To do this we compartmentalized all of the obvious objects involved in the system. These objects included; Study, Settings, Image and a Graphical UI.

Study is a fairly simple object that compares to that of a folder in reality. The folder that Study is intended to represent contains a list of images and obviously has a name. Therefore, Study has a name and contains a list of images. The images are each held in their own class called Image. We implemented a virtual proxy pattern for the image class, a common approach to resource heavy objects. In doing so there are actually two classes for Image, one which is a bare, lightweight class and another which is called upon when the image data is needed. This allows us to have the program function as if it were holding all of the images when the studies are loaded in while saving computer memory because the images are actually only loaded when a study is chosen to be viewed. The actual image class is responsible for communicating with the source of the data to load the image. The image class communicates with the source using another class designed specifically for dealing with reading and writing images to the local network. We separated these read and write tasks to allow for an alternate class to be placed in the design should the customer want to read the images from another source such as a database or a server PC.

Settings is a fairly simple class as well, the settings class is responsible for communicating to the local root folder of the application to store and load settings as necessary. To do this the settings class is aided by another class that is solely responsible for reading and writing files to a directory. We chose this approach to separate the act of dealing with files as a separation of concerns and allow future changes to accept a different data access class that may access the data from another source, be it a database, server PC, anything really, as long as they implement the proper interface included in the design which we will get into later.

The last main, real world, piece we had to implement was a graphical user interface commonly known as a GUI. We were given specific requirements as to how this was supposed to be implemented and we fulfilled all of those requirements. Some of the requirements were that the GUI show two different views, one showing one image and another showing four images in a grid pattern. The GUI also needed the ability to scroll through images forward or backward, obviously the ability to open studies but in addition, be able to save and create new studies. We designed the GUI so that it was as “dumb” as possible, that is that it would know nothing about the classes Study, StudyImage, Settings etc. To do this we used a mediator that we called the ProxyController which we will get into next.

The ProxyController is a class that acts as a mediator between the other classes. The responsibility of the ProxyController is to translate events from the GUI to the other classes. The ProxyController knows that when the GUI sends out an open command, it needs to compile a list of studies and populate a selection window with that list of studies. When a save view command is sent from the GUI the ProxyController knows to pull appropriate information from the GUI and communicate with the Settings class to save the pertinent information from the GUI. To do all of this, the ProxyController is an observer of the GUI, listening to events and reacting as necessary. Because the ProxyController needs to populate a list of studies, this is the class that reads the folder names and creates the Study objects. As with the other classes that dealt with persistent data, this class too uses a separate class specifically designed with reading and writing folders to a specific location, which is currently the local drive.

Finally, we have the DAO collection, I refer to this as a collection rather than a particular class because it is a set of classes but they all perform the same function. Each class involved in this system is responsible for reading and writing data to some persistent model which is currently the local drive. There are three classes, one for dealing with images, another for folders and finally, one for dealing with files for the settings storage. Each class implements and interface to ensure they each have the appropriate methods to function with the system. The interface also allows for abstraction in the classes that use the DAO objects. This way they don’t need to know about the concrete class, rather, they can each accept the same type of object.

Describe High level architecture

Give rationale for major design choices made

Show how the design covers specific requirements

Outline how the design reflects a balance among low coupling, high cohesion, separation of concerns, information hiding, the Law of Demeter, extensibility, reusability, etc. (This can include UML, CRC, Sequence charts etc.

UML Class diagrams

Show main classes and interfaces in your design, along with appropriate relationships

Include cardinality as necessary

DO NOT include state or method information, this is to be captured in CRC cards

Indicate pattern participation using stereotypes (i.e. in guillemets << and >>).

Use multiple class diagrams showing from a top level using subsystems down to a detailed view of subsystems.

Class Responsibilities Collaborators cards

Example template

|  |  |
| --- | --- |
| **Class:** Motor |  |
| **Responsibilities:** The overall responsibility of this class is control of the motor  and prevention of unsafe operation. It provides start/stop control as well as the  ability to move the motor to a fixed angular location. The class can drive the  motor through a predefined time sequence of angular locations. Because this  class often needs to execute in a separate thread, many of its methods are  synchronized. The class is also responsible for providing status on the motor in  terms of current speed and angular position. | |
| **Collaborators** | |
| **Uses:**  SpeedEncoder, MotorPowerSwitch,  EmergencyAlarm, SystemTimer | **Used by:**  MotorThread, InletWaterSupply,  SolarPanelPositioner |
| **Author:** J. Smith | |

Not sure about these because the narrative already explained all of this.

Descriptions of pattern usages

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** Image Proxy | | | **GoF pattern:** Proxy Pattern (Virtual) |
| **Participants** | | | |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** | |
| IStudyImage | Subject | This class defines the necessary methods used in the concrete classes. | |
| ImageProxy | Proxy | This class has a reference to the actual image so that it can be loaded when the appropriate method is called | |
| StudyImage | RealSubject | This class contains that actual data for presenting the image on a display. | |
| **Deviations from the standard pattern:** None | | | |
| **Requirements being covered:**   * A need for a more versatile reference to an object than a simple pointer. * A need to create expensive objects on demand, putting resource heavy tasks off until absolutely necessary. | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** Mediator | | | **GoF pattern:** Mediator |
| **Participants** | | | |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** | |
| ProxyController | aConcreteMediator | This class communicates with various system classes and allows for communication without each class needing to know about the other. | |
| GUI | aColleague | A black box system | |
| Study | aColleague | Responsible for holding a list of images | |
| Settings | aColleague | Responsible for reading and storing images | |
| FolderDAO | aColleague | Responsible for reading and storing folders | |
| **Deviations from the standard pattern:** A Mediator interface was not implemented. | | | |
| **Requirements being covered:**   * A set of objects communicate in a well-defined but complex way * A behavior that is distributed between several classes should be customizable without a lot of subclassing. | | | |
| **Name:** GUI Observations | | | **GoF pattern:** Observer |
| **Participants** | | | |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** | |
| StudyView | Subject | This interface defines the necessary methods to implement a concrete GUI for this system. | |
| MainView | ConcreteSubject | This class implements the methods from the MIVCDisplay interface. | |
| ActionListener | Observer | This interface is a Java native that defines the necessary method to implement to listen to GUI objects. | |
| ProxyController | ConcreteObserver | This class listens to GUI action events and implements the ActionListener to react to do so. | |
| **Deviations from the standard pattern:** None | | | |
| **Requirements being covered:**   * Encapsulating one class’ dependency on another in separate objects to vary and reuse them independently. * Changing one object requires changing the other. * When an object should notify other objects without making assumptions about who they are. | | | |

Sequence Diagrams

for **at least** two non-trivial features in the implementation. See more details on the [second a under design documentations](http://www.se.rit.edu/~swen-262/projects/DesignProjectGuidelines.html)

The current state of the program

and any known issues

OTHER NOTES: This document should be professional with figure and table numbers as well as section numbers and page numbers.