Team MIVC

Project: Medical Image Viewing Console

Geoff Berl, Piper Chester, Colin Ferris, Allen Thomas, Ty Kennedy

Due: 9/30/2013

Table of Contents

1. Design Narrative

As software engineers, we often face the daunting task of building an application that is functional, extensible, closed for alterations and adhering to customer requirements. We must address these issues before we begin code construction. We attempted to design the most simplistic system that met the requirements of the customer, and also allowed for future extensibility. We encapsulated all of the objects involved in the system. These objects include the Study, Settings, Image and a Graphical UI.

A fairly simple object, Study acts in the same way a folder does. 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, Study Image, Settings etc. To do this we used a mediator that we called the Controller which we will get into next.

The Controller is a class that acts as a mediator between the other classes. The responsibility of the Controller is to translate events from the GUI to the other classes. The

Controller 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 Controller 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 Controller is an observer of the GUI, listening to events and reacting as necessary. Because the Controller 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 an 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.

2. Architecture

The controller mediates the logic for the GUI and the system. There is low coupling and the current design is very expandable. The singleton pattern prevents inconsistent settings data across the lifetime of the application.

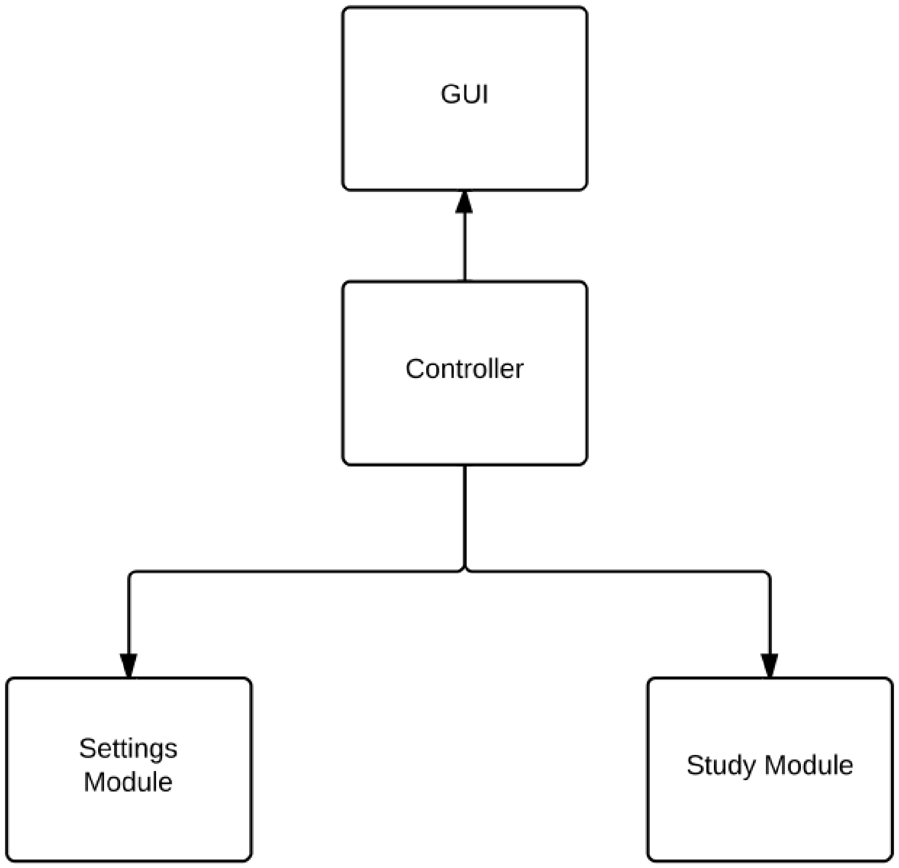


Figure 1. A high level view of the different system components.

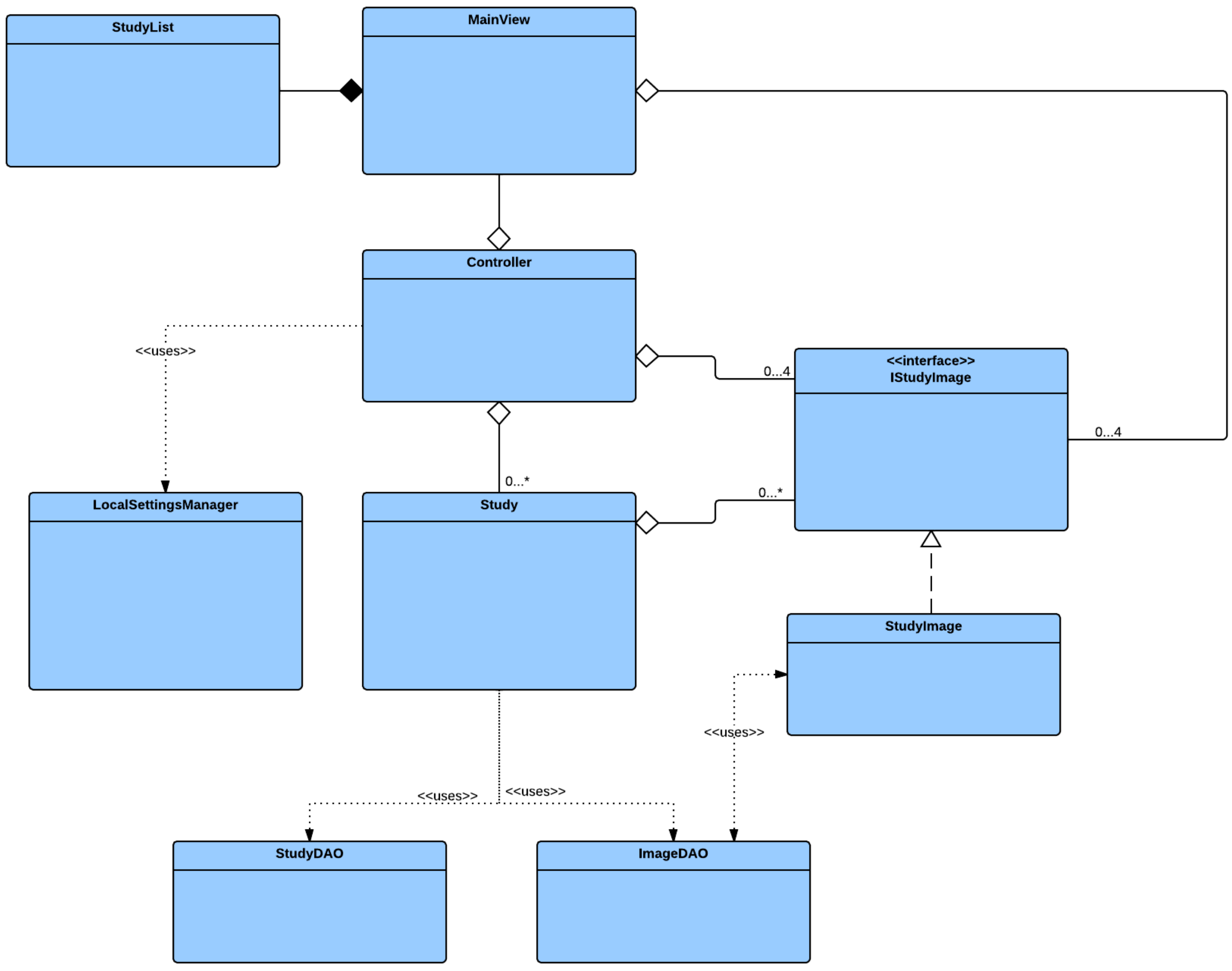


Figure 2. The entire MIVC system.

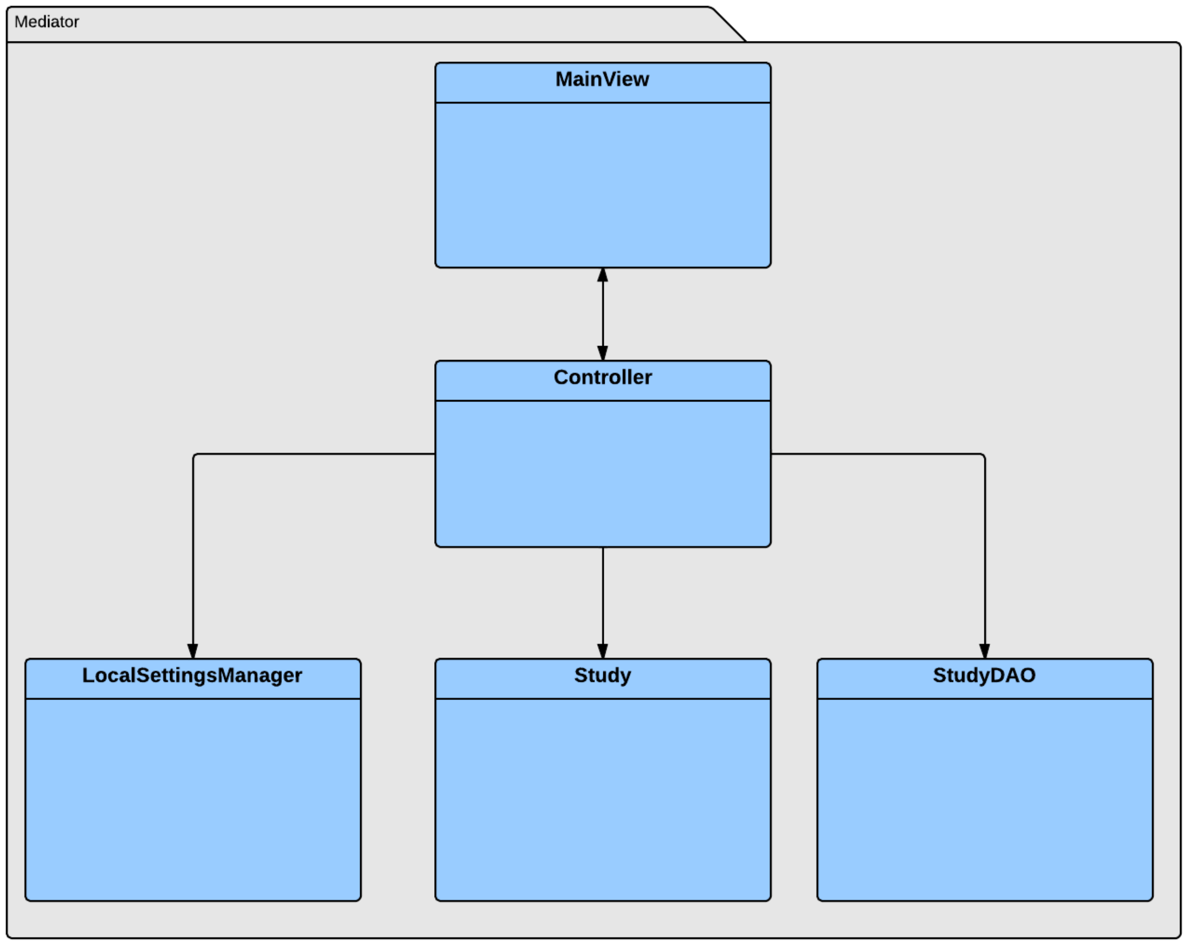


Figure 3. An implementation of the mediator pattern.

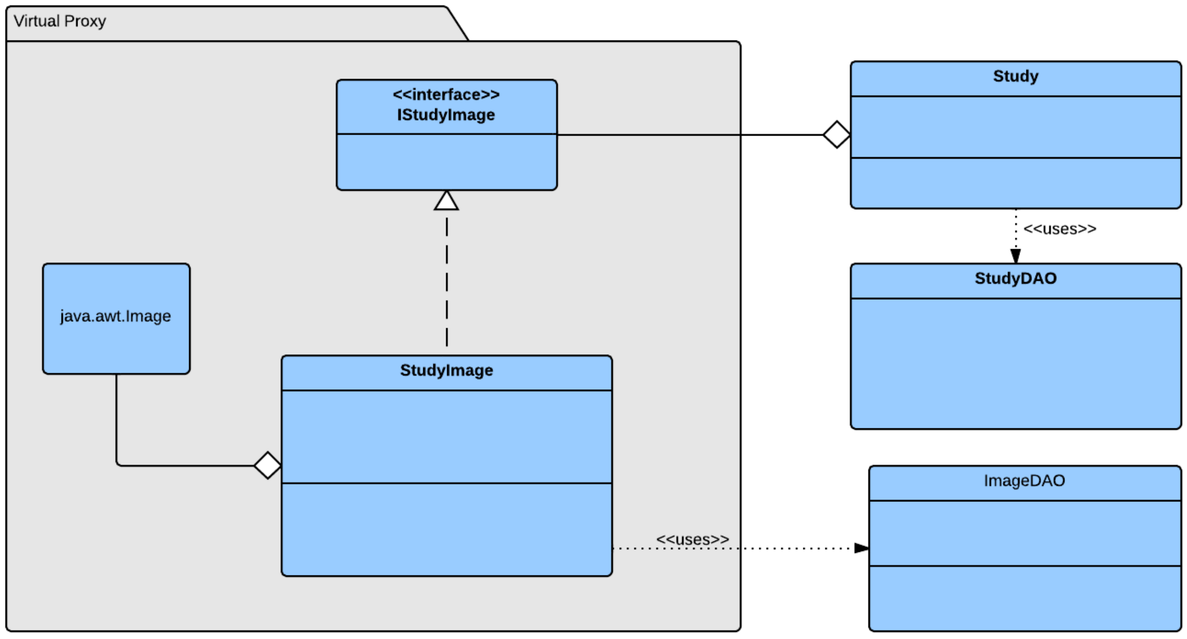


Figure 4. An implementation of the virtual proxy

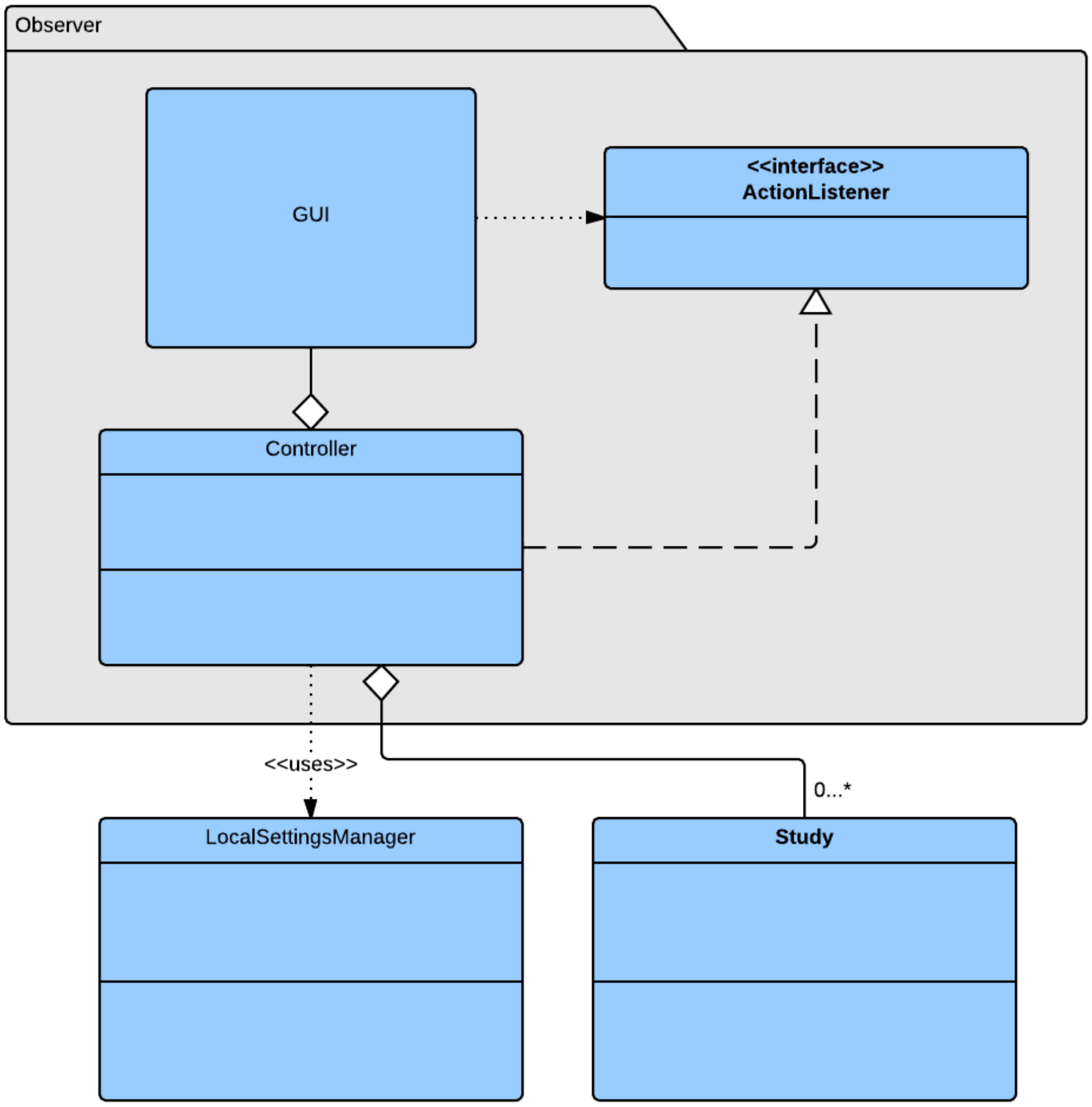


Figure 5. An implementation of the observer pattern.

3. Class Responsibilities and Collaborators

Class: LocalSettingsManager
Responsibilities: Maintains constant data across saves/states
Uses: Saving and loading previous loaded studies, keeping track of variables currentl in use by the system. **Used By:** Controller
Author: Allen Thomas

Class: IStudyImage
Responsibilities: Interface to a study image
Uses: None **Used By:** Study Image
Author: Geoff Berl

Class: Controller
Responsibilities: Handles the logic of the event listeners from the GUI
Uses: Study, StudyView, IStudyImage, LocalSettingsManager, StudyDAO **Used By:** None
Author: Geoff Berl

Class: Study
Responsibilities: Holds images related to the study.
Uses: IStudyImage **Used By:** Controller
Author: Piper Chester

Class: StudyImage
Responsibilities: Implements the IStudyImage interface to become the object that contains Image information about the Study, including image path, image name, etc.
Uses: IStudyImage **Used By:** ImageDAO
Author: Geoff Berl

Class: StudyDAO
Responsibilities: Handles basic IO operations related to studies.
Uses: None **Used By:** Controller
Author: Ty Kennedy

Class: ImageDAO
Responsibilities: Handles basic IO operations related to images.
Uses: IStudyImage **Used By:** Study, StudyImage
Author: Ty Kennedy

Class: QuadView
Responsibilities: Presents a split view of four images from a study to the user.
Uses: IStudyImage **Used By:** MainView
Author: Geoff Berl

Class: MainView
Responsibilities: The main user interface panel of the application.
Uses: Toolbar, SingleView, QuadView, StudyList, StudyView **Used By:** Study, StudyView
Author: Geoff Berl

Class: SingleView
Responsibilities: Presents images from a study to the user.
Uses: IStudyImage **Used By:** MainView
Author: Geoff Berl

Class: StudyList
Responsibilities: List user interface for viewing and selecting studies. The user is able to scroll through multiple studies to select a study that he/she desires.
Uses: None **Used By:** MainView
Author: Geoff Berl

Class: StudyView
Responsibilities: View that displays a single image to the user.
Uses: None **Used By:** MainView
Author: Geoff Berl

Class: Toolbar
Responsibilities: Presents the status of the application, as well as containing all of the various buttons within the application.
Uses: None **Used By:** MainView
Author: Geoff Berl

4. Pattern Usages

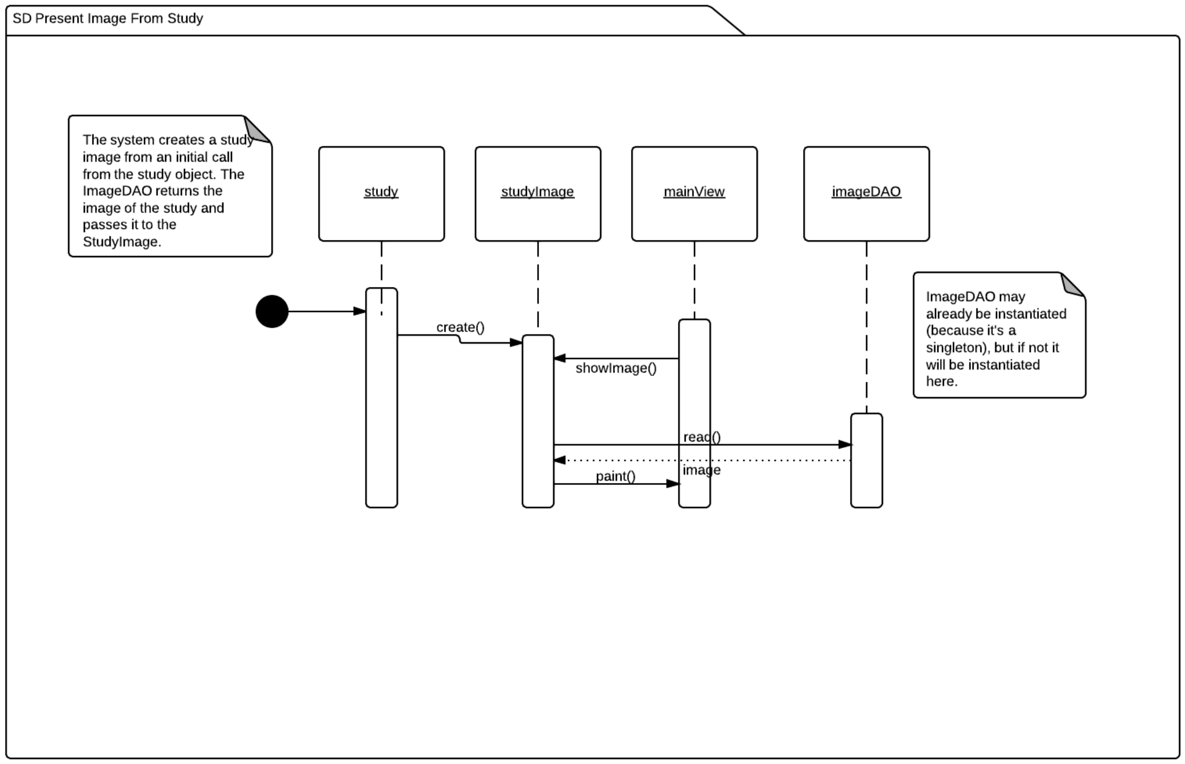
Name: Image Proxy			**GoF pattern:** Proxy Pattern (Virtual)
Participants:			
Class:	**Role in pattern**	**Participant's contribution in the context of the application**	
IStudyImage	Subject	This interface defines the necessary methods used in the concrete classes.	
MainView	Client	This class has a reference to the actual image so that it can be loaded when the appropriate method is called	
StudyImage	Proxy/RealSubject	This class contains that actual data for presenting the image on a display. The actual data is only loaded upon a method call to do so.	
Deviations from the standard pattern: Did not use two classes that implement the interface, the StudyImage class encompasses both functions.			
		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.	

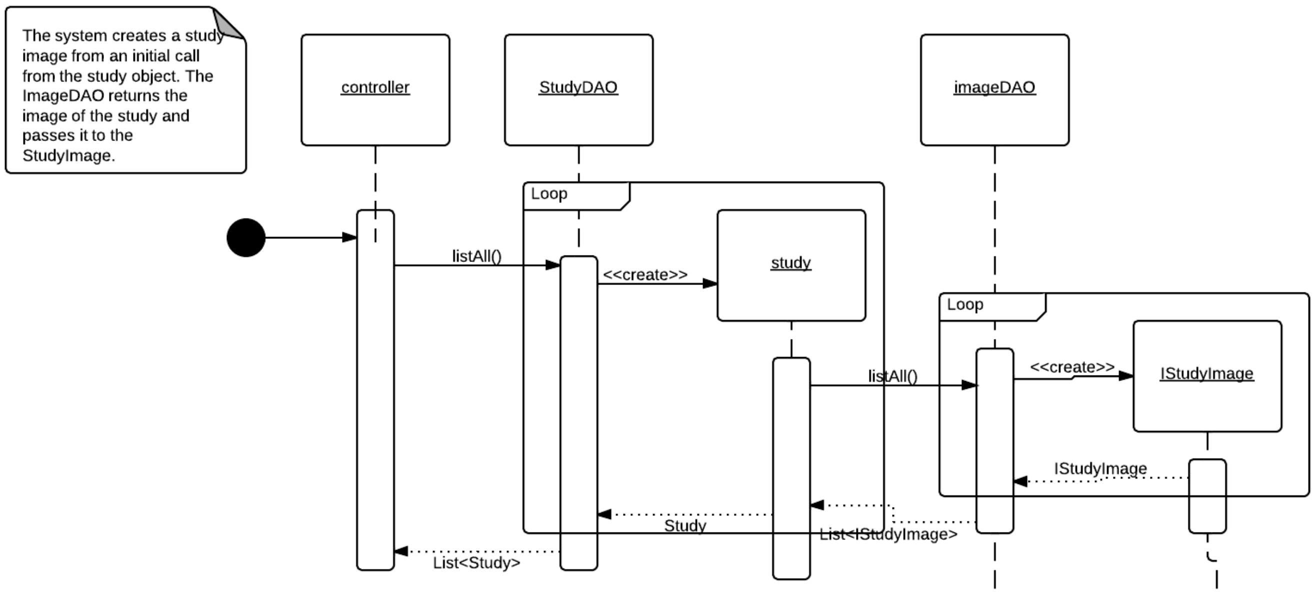
Name: Mediator			**GoF pattern:**

			Mediator
Participants:			
Class:	**Role:**	**Participant's contribution in the context of the application**	
Controller	aConcreteMedi ator	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	
LocalSettingsManager	aColleague	Responsible for reading and storing images	
StudyDAO	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			
Participants			
Class	**Role**	**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.	
Controller	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.	

5. Sequence Diagrams


SD Present Image From Study
The system creates a study image from an initial call from the study object. The ImageDAO returns the image of the study and passes it to the StudyImage.
study
studyImage
mainView
imageDAO
ImageDAO may already be instantiated (because it's a singleton), but if not it will be instantiated here.
create()
showImage()
read()
image
paint()


The system creates a study image from an initial call from the study object. The ImageDAO returns the image of the study and passes it to the StudyImage.
controller
StudyDAO
imageDAO
Loop
listAll()
<<create>>
study
Loop
listAll()
<<create>>
IStudyImage
IStudyImage
Study
List<IStudyImage>
List<Study>

6. Implementation State

Missing Functionality

- If the current view is not saved to file, the warning message does not display.
- The networking has not been implemented for this release.