Medical Image Viewer R1

Design Documentation Summery

Team Petulant-Batman: Dan Rockefeller, Zachary Calfin, Matthew Dennis, Robert Lowe

Submitted: 10/3/2013

Table of Contents

1. Design
2. UML Class Diagrams
3. CRC Card Description
4. Pattern Usage
5. Sequence Diagrams
6. Current State of Implementation

Section 1: Design

The Medical Image Viewing Console’s architecture can be separated into three parts. First, a user interface. Second, a Singleton used to separate the user interface from the back end structure. Third, the back end architecture that implements the actual functionality. This overall structure is shown in Diagram 2 in Section 2.

Each part described can be divided into subsystems as shown in Diagram 2.

* The Study Subsystem manages the creation of an object that represents a study. The Commands Subsystem is responsible for the creation of the list of commands that the User Interface can use.
* The Control Subsystem contains a list of Commands, and a Study, which can be accessed by the User Interface to either get image locations, or get commands that will alter the state of the User Interface.
* The User Interface Subsystem displays the content to the user as defined by the requirements discussed later in this section.
* The final subsystem, is the State. The State machine can be seen by both the User Interface and the Commands, as their behavior relies on the current state of the system.

The first major design choice, to use the Singleton Pattern, was implemented to achieve a low level of coupling throughout the system. All coupling is handled by this Singleton. In most cases, it is difficult to avoid “The Blob” anti-pattern while using the Singleton Pattern. In the case of this implementation, only bare functionality is implemented in this class, therefore avoiding the anti-pattern. The Singleton’s main purpose is to coordinate actions across a system with one object. A high level of cohesion was achieved by implementing only the functionality to coordinate actions, not the actions themselves. This pattern is also essential to information hiding in the system since it separates the User Interface from the system’s design.

The study system mechanics was the second major design choice, leading to very high extensibility. A study factory Interface allows the implementation of different types of studies. Currently there are two different types of studies outlined. First, a normal study, and second a remote study, which has yet to be implemented. These studies are stored in the Singleton labeled Director as seen in Diagram 1 in Section 2. During the very late stages of the design process, it was revealed that the main concern for creating studies was creating a polymorphic subsystem, instead of solving the problem of an over-complex representation. The original builder pattern was changed to a factory pattern to fit the needs of the project. The study class created is the representation of the study group outlined in the requirements. The factory method used for creating study allows for the addition of other types of studies, such as the remote study, outlined in the requirements.

The third major design choice was to implement a state machine. There are two main reasons the state pattern was implemented. First, it allowed the GUI to know what display mode it should be in. Second, it allowed for extremely efficient commands. Diagram 1 in Section 2 shows how the State can be seen by the various commands that the GUI uses to perform its various duties. Each command would need to be re-created in the event of a state change, however, by changing the way a command acts depending on the current state, the command objects only needs to be created once when starting the program. This state machine is the implementation of the requirement specifying multiple display modes. It also offers an easy way to save the display state as required by another requirement.

Lastly, The Command system was created to remove functionality from the GUI and encapsulate it into individual command classes leading to a more cohesive product, with the additional benefit of extensibility if more commands need to be added. These commands are created during the creation of the Director Singleton, and stored in an array. The way each of these commands act depends on the state of the system. This pattern also implements the concept of information hiding as it separates the command process from the GUI object. The command system is the implementation of the requirements specifying the need to navigate throughout the study in groups of fours or one by one.

Section 2. UML Class Diagrams

Diagram 1: Master Diagram

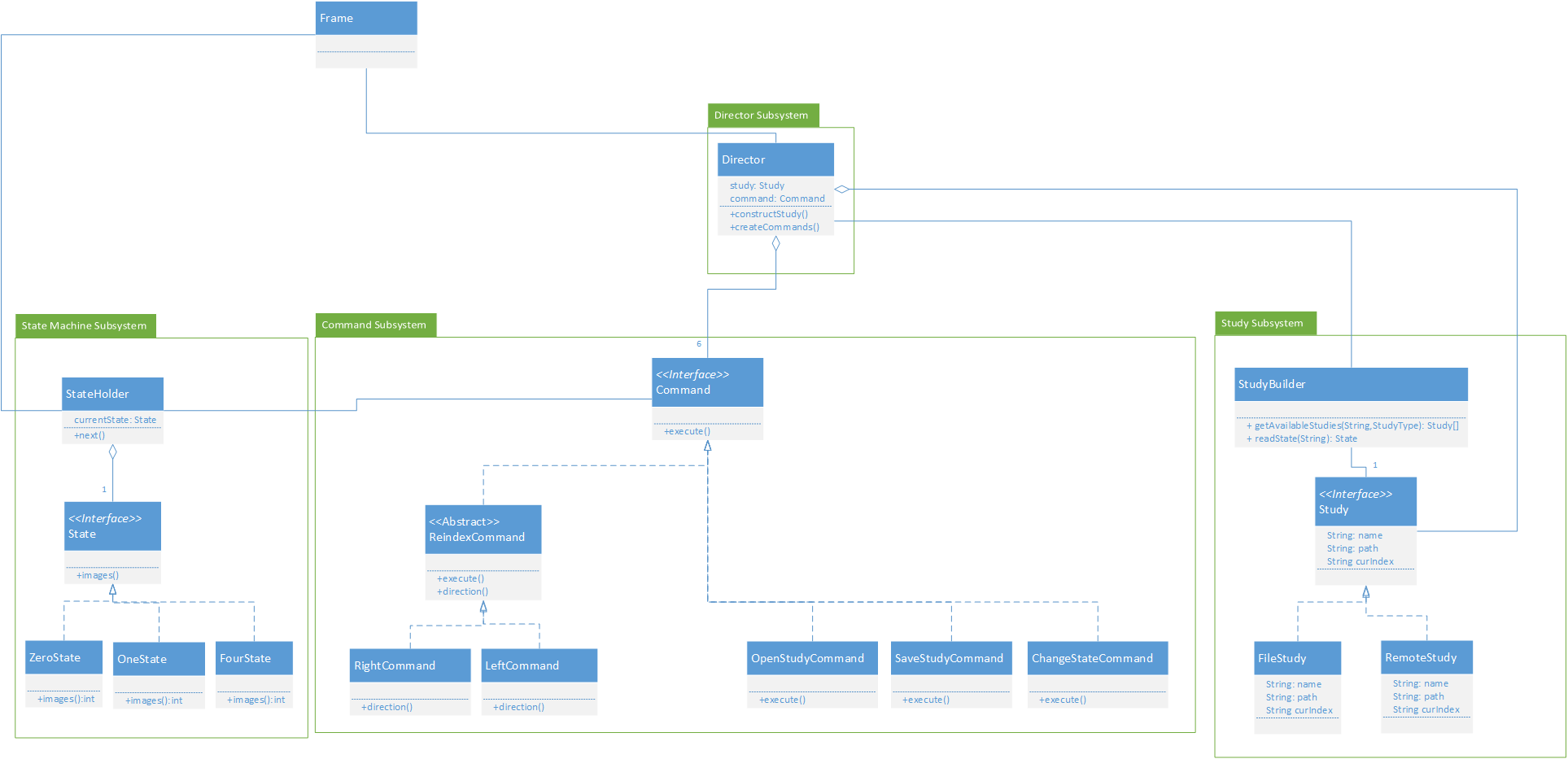


Diagram 2: Structure Diagram

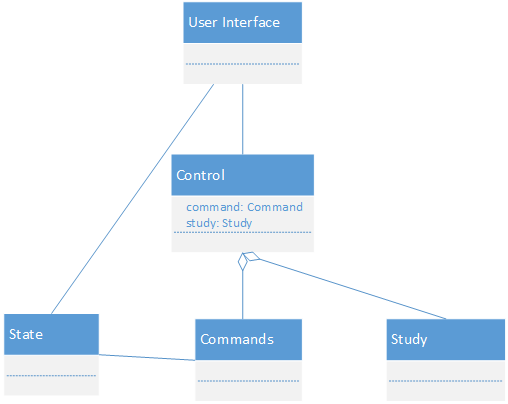


Diagram 3: State Diagram

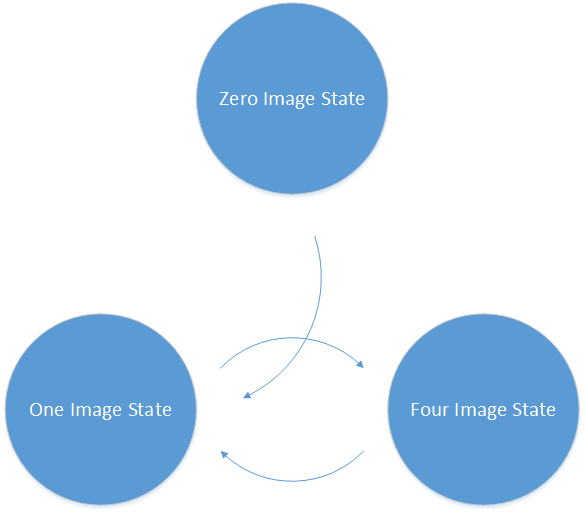
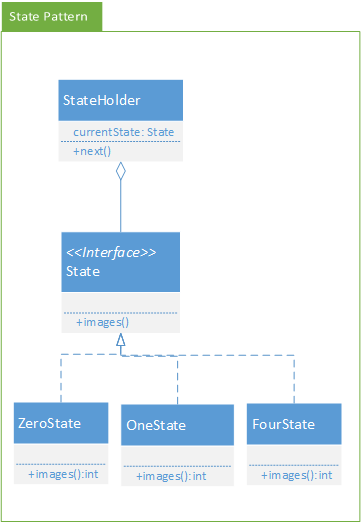
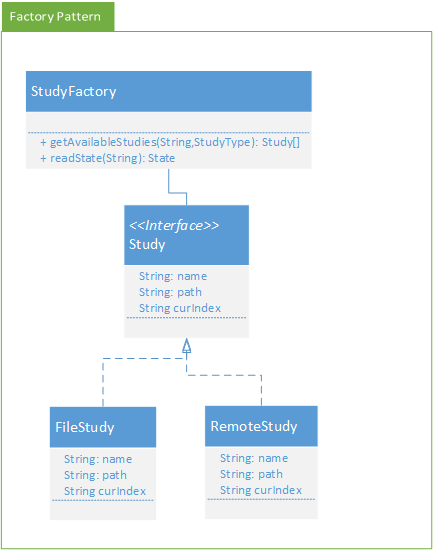
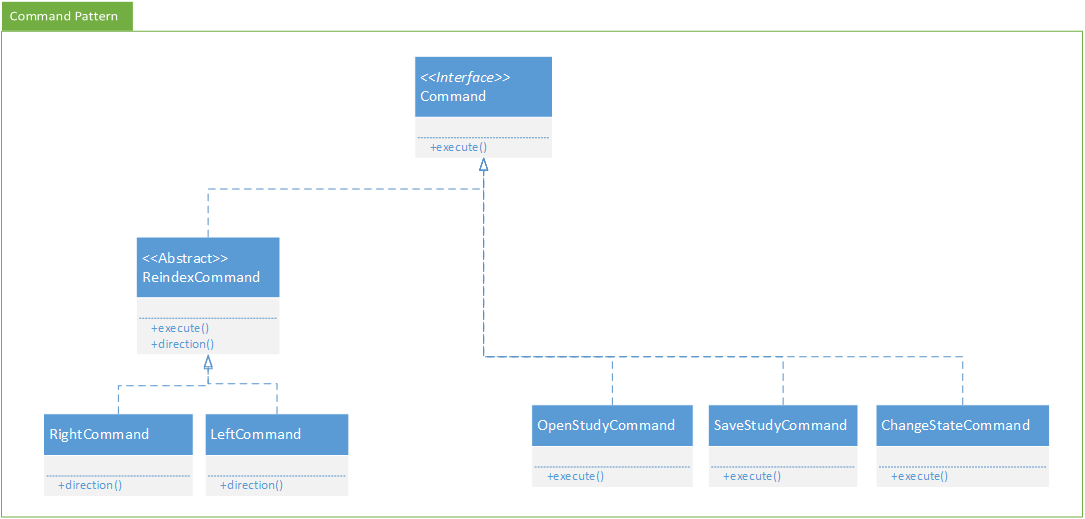
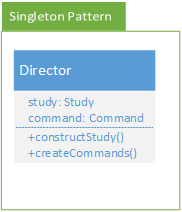


Diagram 4: Pattern Usage





Section 4: CRC card descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** StateHolder | |  | |
| **Responsibilities:** Holds the current State the system is in | | | |
| **Collaborators** | | | |
| **Uses:** ZeroState,OneState,FourState | | **Used by:** All commands, GUI | |
| **Author:** Robert Lowe | | | |
| **Class:** State | |  | |
| **Responsibilities:** Store system state in a class | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** StateHolder | |
| **Author:** Robert Lowe | | | |
| **Class:** ZeroState | |  | |
| **Responsibilities:** Returns a number indicating how many images to display to the screen | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** StateHolder | |
| **Author:** Robert Lowe | | | |
| **Class:** OneState | |  | |
| **Responsibilities:** Returns a number indicating how many images to display to the screen | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** StateHolder | |
| **Author:** Robert Lowe | | | |
| **Class:** FourState | |  | |
| **Responsibilities:** Returns a number indicating how many images to display to the screen | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** StateHolder | |
| **Author:** Robert Lowe | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** Command | |  | |
| **Responsibilities:** Stubs out functionality for commands | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** ConcreteCommand | |
| **Author:** Robert Lowe | | | |
| **Class:** ChangeModeCommand | |  | |
| **Responsibilities:** Changing the mode of the state machine | | | |
| **Collaborators** | | | |
| **Uses:** State | | **Used by:** GUI | |
| **Author:** Robert Lowe | | | |
| **Class:** LeftCommand | |  | |
| **Responsibilities:** Change the state of the currently loaded Study to reflect a move to the left | | | |
| **Collaborators** | | | |
| **Uses:** Study, Director | | **Used by:** GUI | |
| **Author:** Robert Lowe | | | |
| **Class:** OpenStudyCommand | |  | |
| **Responsibilities:** Opens a new study and stored it in director | | | |
| **Collaborators** | | | |
| **Uses:** Director, Study, StudyBuilder | | **Used by:** GUI | |
| **Author:** Robert Lowe | | | |
| **Class:** RightCommand | |  | |
| **Responsibilities:** Change the state of the currently loaded Study to reflect a move to the right | | | |
| **Collaborators** | | | |
| **Uses:** Study, Director | | **Used by:** GUI | |
| **Author:** Robert Lowe | | | |

|  |  |
| --- | --- |
| **Class:** SaveStudyCommand |  |
| **Responsibilities:**  Saving the state of the Study to file | |
| **Collaborators** | |
| **Uses:** Study | **Used by:** GUI |
| **Author:** Robert Lowe | |

|  |  |
| --- | --- |
| **Class:** Director |  |
| **Responsibilities:** Simplifys talking to and stores the currently opened Study | |
| **Collaborators** | |
| **Uses:** Study, State | **Used by:** GUI, Commands |
| **Author:** Robert Lowe | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Class:** FileStudy | |  | |
| **Responsibilities:** Store the state of a file based study and methods for writing and reading the filestudy to the filesystem | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** StudyBuilder, Director, Commands | |
| **Author:** Matthew Dennis | | | |
| **Class:** RemoteStudy | |  | |
| **Responsibilities:** Dummy class for unimplemented Remote Study Type | | | |
| **Collaborators** | | | |
| **Uses:** | | **Used by:** StudyFactory | |
| **Author:** Matthew Dennis | | | |
| **Class:** StudyFactory | |  | |
| **Responsibilities:** Creates a Study based on an input | | | |
| **Collaborators** | | | |
| **Uses:** Study | | **Used by:** Commands | |
| **Author:** Matthew Dennis | | | |

|  |  |
| --- | --- |
| **Class:** Frame |  |
| **Responsibilities:** Displays Images, User Interface | |
| **Collaborators** | |
| **Uses:** StateHolder, Director | **Used by:** |
| **Author:** Zachary Calfin | |

Section 5: Pattern Usage

1. Study Factory Pattern

Director: Creator that calls Study Factory to create a Study

The Director is responsible for creating a study that holds information that the GUI uses to display pictures at a specific file path.

StudyFactory: Interface that creates either a FileStudy or a RemoteStudy

The StudyFactory allows there to be multiple types of studies, since it is relatively easy to add more subclasses.

FileStudy: Concrete Product and object representation of a study as defined in the requirements

The file study maintains all the information the GUI needs to know about the current image(s) being displayed, and the images that can be displayed.

RemoteStudy: Concrete Product and object representation of a remote study as defined in the requirements

The Remote study has the same responsibilities as a FileStudy but accesses the files from a remote location

1. UI Action Commands Pattern

Director: Invoker that holds command objects

Responsible for holding the commands that the UI calls

Command: Abstract interface that creates a command

The Study Command class is an abstract class that creates multiple function commands. Commands can be added due to the nature of the Command pattern.

ChangeStateCommand: Concrete command that changes the display mode

Responsible for changing the display mode in the UI and triggering a state change in the state machine.

NewStudyCommand: Concrete command that makes a new study

Creates a new study which images can be added to.

LeftCommand: Concrete command that increments up 1 or 4 pictures

Indexes the pictures in the GUI by 1 or by 4 depending on the display mode.

RightCommand: Concrete command that increments down 1 or 4 pictures

Increments the pictures in the GUI down by 1 or 4 depending on the display mode.

SaveStudyCommand: Concrete command that saves a study

Saves the study in its current state so when it’s opened, it will open on the picture and state that it was closed on.

OpenStudyCommand: Concrete command that opens a study

Opens a specific study that the user specified and changes the state

1. Director Singleton Pattern

Director: responsible for containing a Study and Commands

Holds the study which can be accessed by the GUI to get file locations. It also holds commands that can be called by the GUI to change its status.

1. State Singleton Pattern

StateHolder: Responsible for containing a state

Holds the state in a singleton so it can be easily accessed by the User Interface or the Commands

1. UI State Pattern

StateHolder: Responsible for containing a state

Holds the state in a singleton so it can be easily accessed by the User Interface or the Commands

State: Abstract interface that creates a state

Interface that is a basic template for the different states that can be created. It offers the ability to add other states if needed.

ZeroState: Concrete state for a GUI containing no pictures

Concrete initial state when the program opens up for displaying a blank page

OneState: Concrete state for a GUI containing one picture

Concrete state in which the GUI displays only one picture at a time

FourState: Concrete state for a GUI containing four pictures

Concrete state in which the GUI displays four pictures at a time

Section 5: Sequence Diagrams

Diagram 1: Opening a Study

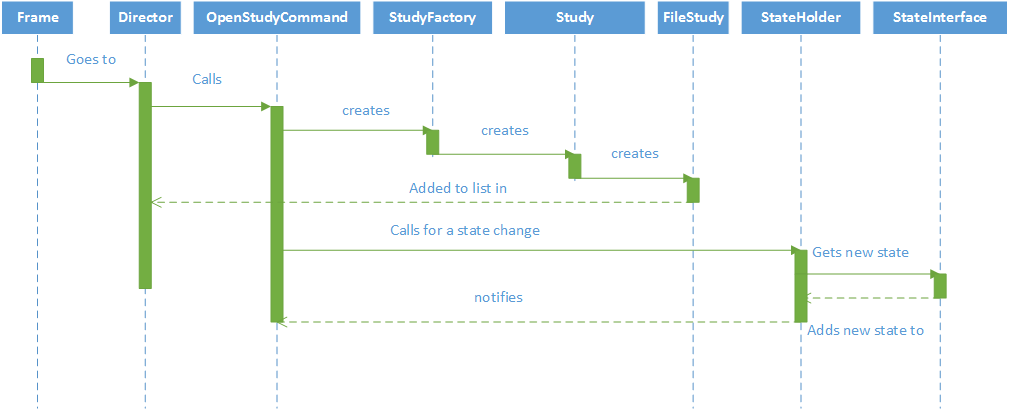
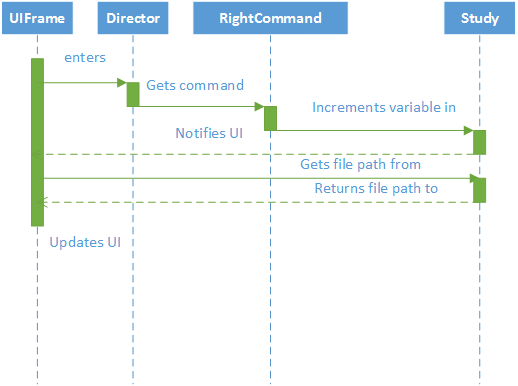


Diagram 2: Moving Right



Section 6: Current State of Implementation

The implementation of the featured design is in a working but with errors. Currently the implementation has full functionality with a few errors.