Brendan Kirby

Michael Surdouski

Artur Braga

William Dougherty

Radiologists

Medical Image Viewer Release 1

10/01/13

**Section 1 - Narrative**

--------------------------------------------------------------------------------------------------------------------

**Overall Structure (Refer to Figure 1.1)**

The overarching design choice for the medical image viewer is that of Model-View-Controller. Model-View-Controller is a good design choice for this because it requires user interaction with the system in a pre-scripted way [scrolling, image transversing, display choice, etc.].

**Model (Refer to Figure 1.3)**

Our model consists of multiple classes which define the structure and lower level logic of the medical image viewer:

* Session: This class holds a reference to the logic to change state, the state which the view should reflect, the current paths for any current images being displayed, and a reference to the mouse for system interaction such as scrolling. It does not define much behavior, rather it is just a holder and state for the system.
* Collection: There is a 1 to 1 aggregation to this class from Session. Defines the behavior for image and display manipulation. This class does not know how it retrieves or loads images, it just makes the choices to do them. It holds a CollectionPath reference which holds the list of paths to be doing the scrolling through.

The model makes use of 3 different design patterns:

* Strategy: To access the paths where files may be located changes depending on the type of place you wish to access them from. The strategy design pattern makes it easy to assemble a list of paths, whether or not they are accessed online through a website or through a local file on your computer. By using this pattern, we efficiently eliminate any problems arising from future requirement changes and/or additions.
* Observer: Changes in the Session are observed for reporting to the view. This is a standard choice for MVC architecture to decrease coupling within the system.

Some things to note:

* The strategy pattern is a way to decrease coupling in the creation of a list of paths. Any changes or additions in the way paths are retrieved is now easily implemented with an addition of a new class which implements the path interface. Therefore, there would be little, if any, changing of previous code to implement this new feature.
* It might have different values based on the state that it is in, but the operations performed on it will be concurrent either way.
* There is a 1 to 1 aggregation of Collection to Session. Collection separates the lower level logic away from the state which Session describes.
* There is a many to 1 aggregation from Image to Collection.
* There is a 1 to 1 aggregation from CollectionPath to Collection.

**Controller (Refer to figure 1.2)**

The controller contains the higher level logic (logic needed to use the view), the running of the program, and the references to the model and view:

* Main: Allows access to main controller, handles termination, and gives a level of abstraction from the user.
* Main Controller: Holds a reference to Session inside the model, Main View inside the view, and contains action logic for commands given from the user.
* Actions: An interface which all user commands must use. Because of potential requirement changes and as to decrease coupling, therefore increasing cohesion, all commands must extend the actions interface. If any new commands are to be used, it would be very easy to implement them because main control can call them in the same way. Without this interface, a new class would have to be made which would then have to be implemented in a separate way inside of Main Controller.

Some things to note:

* There is a 1 to 1 aggregation from Session to Main Controller.
* There is a 1 to 1 aggregation from Main View to Main Controller.
* Main Controller is static inside of Main because Main Controller has no state to change.

**View (Refer to Figure 1.4)**

The view handles any visualization which needs to be created for the person using the medical image viewer. It contains listeners for button clicks as well as the logic to create displays for windows:

* Main View: Primary display used to fill the viewers display. Contains all buttons, windows, menus, and has an aggregation to Main Controller. Does not handle any logic.
* Window Factory: Creates window views based on the display state and the current image(s) being rendered on screen.
* Window: The images being displayed on screen is described in an object implementing this interface. The logic for the changes in display state and image state is also contained within this object.

The view makes use of 2 design patterns:

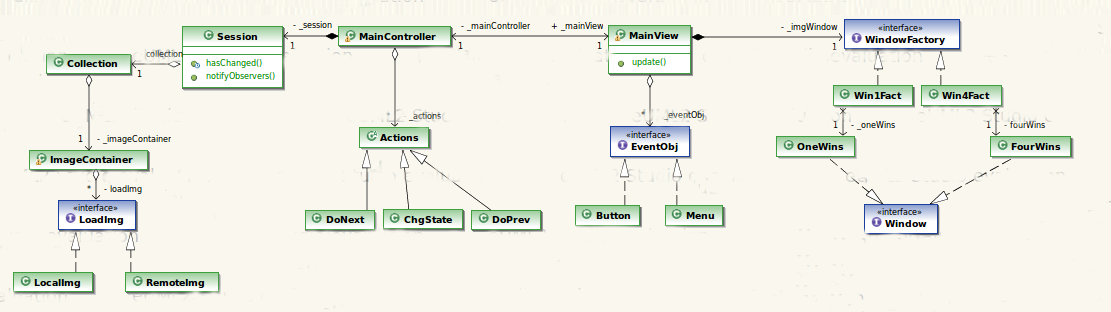
* Observer: As described before, the view needs to update depending on changes in Session in the model.
* Factory: The factory design pattern is useful here for a couple of reasons. A window doesn’t need separate commands depending on the type of window. However, the commands require different behavior depending on the command. Therefore, the window factory allows us to create lots of windows with generic commands but with independent behavior. Also, the windows factory allows the creation of windows to not only be abstracted from the window, but the Main View as well. This allows us to implement a new display for the window without any hassle if the requirements change.

Some things to note:

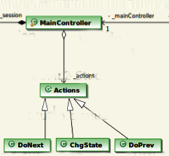
* There is a 1 to 1 aggregation from the Window Factory to the Main View.
* There is a 1 to finite amount x for the buttons contained within the Main View.
* The window factory’s behavior also allows us to implement new features that the medical image viewer must have if the windows all have the same features. A change which affects views different might create the need for change within the system.\*\*\*
* Window extends JPanel

**Section 2 - UML Class diagrams**

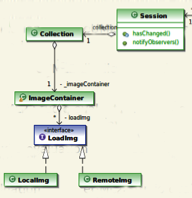
--------------------------------------------------------------------------------------------------------------------



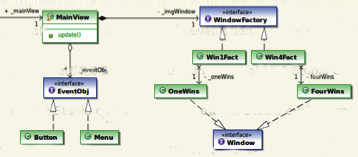
**Figure 1.1 - Model-View-Controller Design Pattern**



**Figure 1.2 - Controller**



**Figure 1.3 - Model**



**Figure 1.4 - View**

**Section 3 - CRC Descriptions**

--------------------------------------------------------------------------------------------------------------------

|  |  |
| --- | --- |
| **Class: Main** |  |
| **Responsibilities:** Allows access to main controller and handles termination |  |
| **Collaborators** |  |
| **Uses: MainController** | **Used by:** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: Main Controller** |  |
| **Responsibilities:** Holds a reference to Session inside the model and Main View inside the view. Contains action logic for commands given from the user, such as scrolling or changing display state. |  |
| **Collaborators** |  |
| **Uses: MainView, Session, Actions** | **Used by: Main** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: Actions** |  |
| **Responsibilities:** An interface which all user commands must use. Because of potential requirement changes and as to decrease coupling, therefore increasing cohesion, all commands must implement the actions interface. |  |
| **Collaborators** |  |
| **Uses: MainController** | **Used by: DoNext, DoPrev, ChangeState** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: DoNext** |  |
| **Responsibilities:** Scroll to the next image or images, depending on the display state |  |
| **Collaborators** |  |
| **Uses: Actions** | **Used by:** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: DoPrev** |  |
| **Responsibilities:** Scroll to the previous image or images, depending on the display state |  |
| **Collaborators** |  |
| **Uses: Actions** | **Used by:** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: ChangeState** |  |
| **Responsibilities:** Changes the display state to either one image or four tiled images |  |
| **Collaborators** |  |
| **Uses: Actions** | **Used by:** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: MainView** |  |
| **Responsibilities:** Primary display used to fill the viewers display. Contains all buttons, windows, menus, and has an aggregation to Main Controller. This class does not handle any logic. |  |
| **Collaborators** |  |
| **Uses: WindowFactory** | **Used by: MainController** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: WindowFactory** |  |
| **Responsibilities:** Creates window views based on the display state and the current images being rendered on screen. |  |
| **Collaborators** |  |
| **Uses:** | **Used by: Window** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: Window** |  |
| **Responsibilities:** Extends the JPanel interface. The images being displayed on screen are described in an object implementing this interface. The logic for the changes in display state and image state is also contained within this object. |  |
| **Collaborators** |  |
| **Uses: WindowFactory** | **Used by:** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: Session** |  |
| **Responsibilities:** This class holds a reference to the logic to change state, the state which the view should reflect, the current paths for any current images being displayed, and a reference to the mouse for system interaction such as scrolling. It does not define much behavior, rather it is just a holder and state for the system. |  |
| **Collaborators** |  |
| **Uses: Collection** | **Used by: MainController** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: Collection** |  |
| **Responsibilities:** Defines the behavior for image and display manipulation. This class does not know how it retrieves or loads images, it just makes the choices to do them. It holds a CollectionPath reference which holds the list of paths to be doing the scrolling through |  |
| **Collaborators** |  |
| **Uses: Image, CollectionPath** | **Used by: Session** |
| **Author: Brendan, Michael, Artur, Will** |  |

|  |  |
| --- | --- |
| **Class: CollectionPath** |  |
| **Responsibilities:** Represents a list of paths to scroll through images. |  |
| **Collaborators** |  |
| **Uses:** | **Used by: Collection** |
| **Author: Brendan, Michael, Artur, Will** |  |

**Section 4 - Pattern Usage**

--------------------------------------------------------------------------------------------------------------------

|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Window Factory** |  |  | **GoF pattern: Factory method** |
| **Participants** |  |  |  |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** |  |
| **WindowFactory** | **Concrete Class** | This class acts as the set up for the various types of windows you might need to display the various images in the GUI. |  |
| **Window** | **Interface** | The images being displayed on screen go through this class, allows a new display to be created without too much hassle |  |
| **Deviations from the standard pattern: None** |  |  |  |
| **Requirements being covered:**  · The system shall display one study at a time.  · The system shall allow the user to view four images from the study tiled two images by two images. |  |  |  |

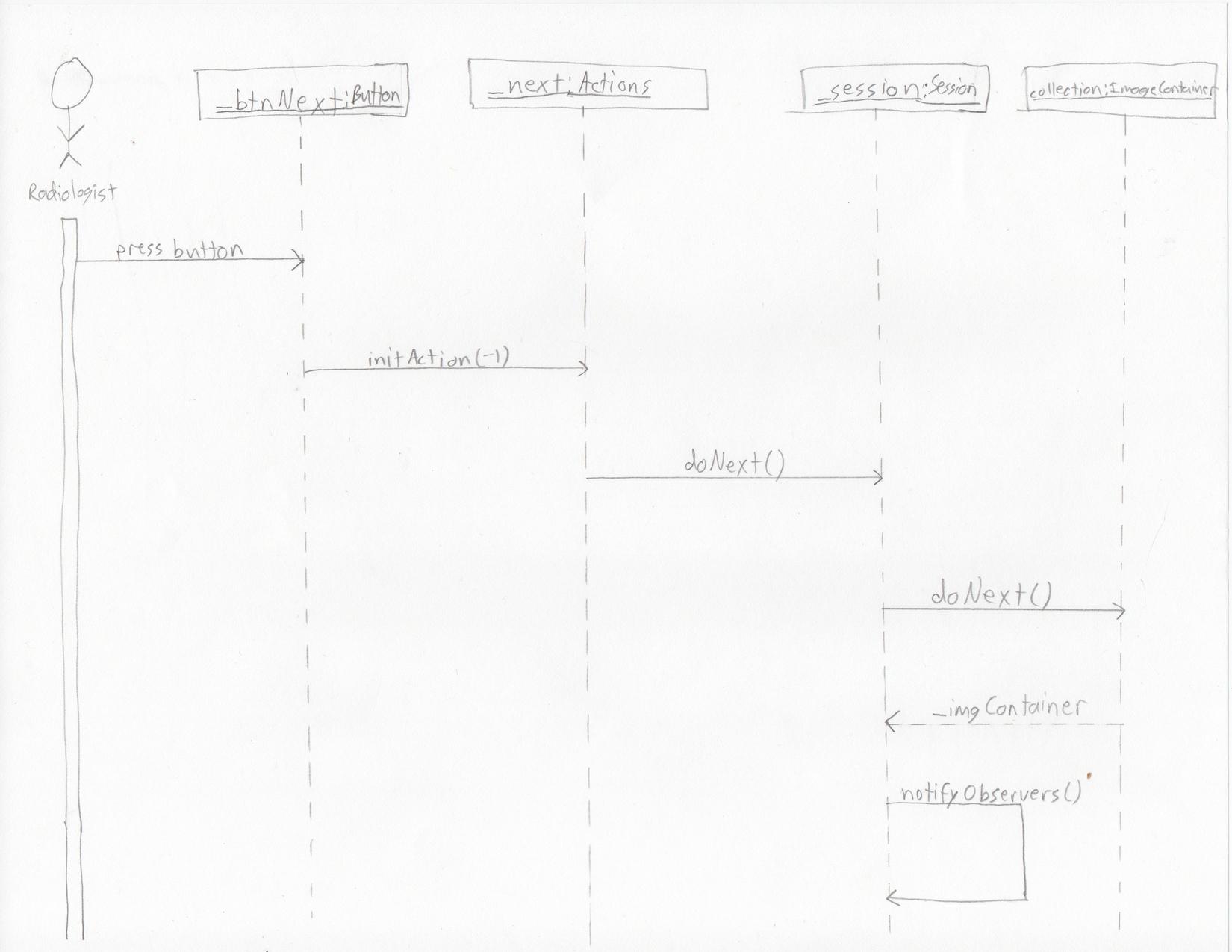
|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Observer** |  |  | **GoF pattern: Observer method** |
| **Participants** |  |  |  |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** |  |
| **Session** | **Concrete Class** | Holds a reference to the logic to change state as well as any current paths for any current images being displayed and a reference to the mouse for system interaction. |  |
| **MainView** | **Interface** | Observes the session and updates depending on how it changes |  |
| **Deviations from the standard pattern: None** |  |  |  |
| **Requirements being covered:** Changes in the Session are observed for reporting to the view |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Image Path** |  |  | **GoF pattern: Strategy Method** |
| **Participants** |  |  |  |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** |  |
| **Collection** | **Concrete Class** | Defines the behavior for image and display manipulation |  |
| **CollectionPath** | **Concrete Class** | Holds a list of paths that can be scrolled though |  |
| **Deviations from the standard pattern: None** |  |  |  |
| **Requirements being covered:** The system shall provide display modes for viewing images within the currently selected study. |  |  |  |

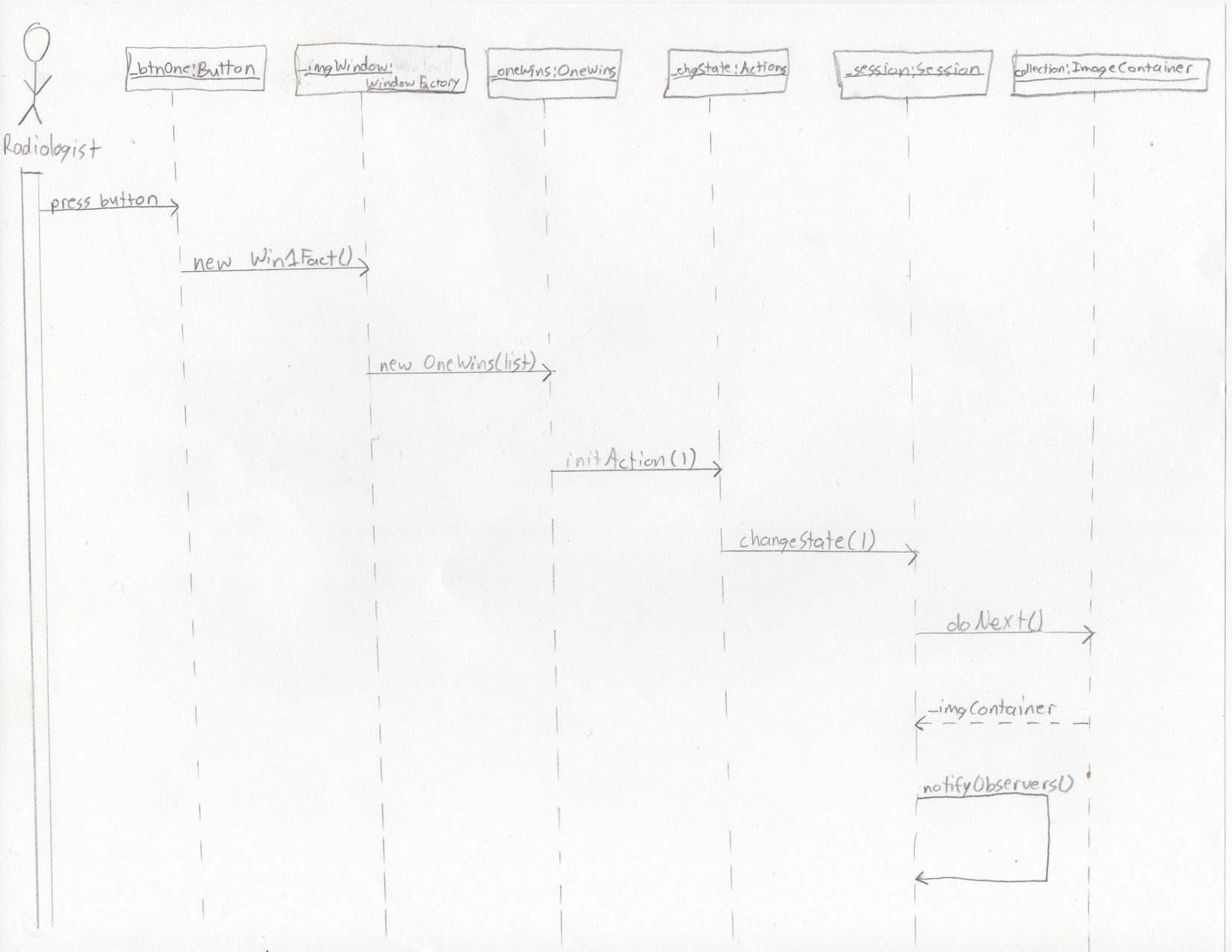
|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Images** |  |  | **GoF pattern: Flyweight Method** |
| **Participants** |  |  |  |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** |  |
| **Collection** | **Concrete Class** | Holds a collection of images that can be displayed |  |
| **Deviations from the standard pattern: None** |  |  |  |
| **Requirements being covered:**  · The system shall read medical images stored as JPEG images.  · The system shall work with images in groupings known as *studies.* |  |  |  |

**Section 5 - Sequence Diagrams**

--------------------------------------------------------------------------------------------------------------------



**Figure 2.1 - Scroll to next image(s)**



**Figure 2.2 - Change display state to one image**

**Feature: Scroll next (Refer to Figure 2.1)**

1. User presses the Next button in the Window
2. initAction(-1) is called on \_next
3. doNext() is called on \_session
4. doNext() is called on collection, which is an ImageContainer
5. collections returns \_imgContainer to \_session
6. \_session() calls notifyObservers()

**Feature: Change display state to one image (Refer to Figure 2.2)**

1. User presses One button in the Window
2. A new Win1Fact named \_imgWindow is created
3. A new OneWins named \_oneWins is created with a list of images
4. initAction(1) is called on \_chgState
5. changeState(1) is called on \_session
6. doNext() is called on collection, which is an ImageContainer
7. collections returns \_imgContainer to \_session
8. \_session() calls notifyObservers()

**Section 6 - State of Implementation**

--------------------------------------------------------------------------------------------------------------------