THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

DEPARTMENT OF COMPUTING

COMP3211 Software Engineering

Project Group11

Personal Information Manager

Design Document

Name:

LIU Minghao

YE Haowen

ZHANG Tianyi

Student ID:

1 PIM Achitecture

1.1 MVC

The Model-View-Controller (MVC) separates the presentation from the interaction from the system data. The Personal Information Management is divided into three mutually logical building blocks. The Model component handles all potential logics which manages all data and accepts actions that transmit user commands/requests. The View component defines the Command Line View and the updated views responds from Model. The Controller component manages the user interface, such as selecting the crossponding operations, and these interactions are transmitted to the view and model. The MVC structure is shown in Figure 1. The details of MVC is shown in Figure 2.

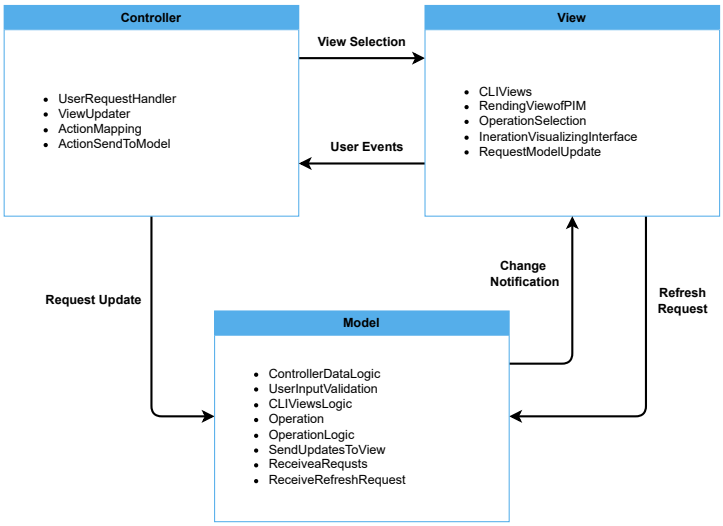

Figure 1: Structure of MVC

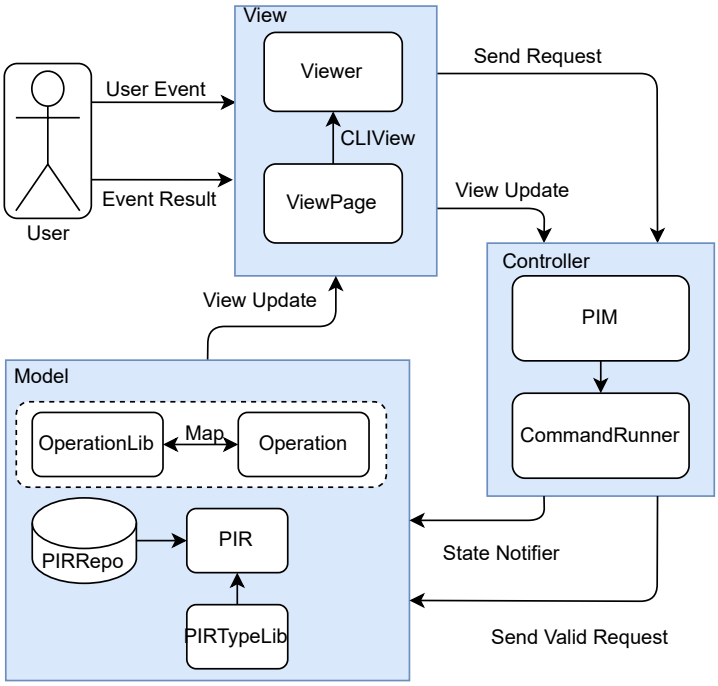

Figure 2: Details of MVC

1.2 Reasons to use MVC

> - There are multiple ways to view and interact with data
> - Future requirements for interaction and presentation of data are unkonwn

1.2.1 Reliability

- Since the *Model* part for function of the system is separate from other parts, *Model* can handle data without depending on how data is presented or business rules are applied.
- This is crucial for a PIM, as it handles a variety of data types that need to be reliably managed and persisted.

1.2.2 Flexibility

- The functional *Model* part is separate from , which brings a lot of flexibility to the *Controller* and *View* part.
- For *Controller* of PIM, PIMs typically have complex user interactions and a variety of business rules. The *Controller* in MVC can manage these interactions, keeping the user interface and data model separate.
- For *View* of PIM, PIMs often require different views for calendars, or list views for contacts. The *View* in MVC can provide these different presentations of data without changing the underlying data structure and handling methods.

1.2.3 Extensibility

- As users' PIM needs grow, new features may need to be added to the PIM software. MVC allows new functionalities to be added with minimal disruption to the existing system, as modifyication or addition Models, Views, or Controllers is independent.

1.2.4 Easy to test and maintain

- MVC's separation of concerns makes it easier to test and maintain business logic of *Controller*, functionalities and data handling logic of *Model* and user interfaces of *View* independently, leading to more stable releases.

1.2.5 Support for different platforms

- MVC supports a consistent user experience across different platforms. For example, whether accessing the PIM via a web browser or a native app, the Model and *Controller* logic can stay the same, while only the View adapts to the specific platform.

1.3 How the components of the MVC were instantiated in the context of the PIM

1.3.1 Model

> - The Modle component manages the system data and associated operations on the data

- The Model represents user data like PIRs with their properties, PIR types. Besides, model will handle all the functionality such as creation, modification, printing, deletion, storing and loading.

1.3.2 View

> - The View component defines and manages how the data is presented to the user

- The View stores different View Pages template (menu, error message, PIR list, ets.) as well as methods to generate proper View Page with filling information.

1.3.3 Controller

> - The Controller component manages user interaction and parses these interaction to the View and the Model

- The Controller handles operation process and user interaction, such as the interaction process to list a menu, let user to input description, time or other fields of PIRs. Then send those information to Model to handle operation or to View to build a View Page together with View Page templates.

2 Major Code Component

Both the structure of and the relationship among the major code components are shown in UML diagrams, which follows the Model-View-Controller structure.

2.1 Class Diagram

2.1.1 Model

The model is the core of the PIM, which is shown in Figure 3 and Figure 3.

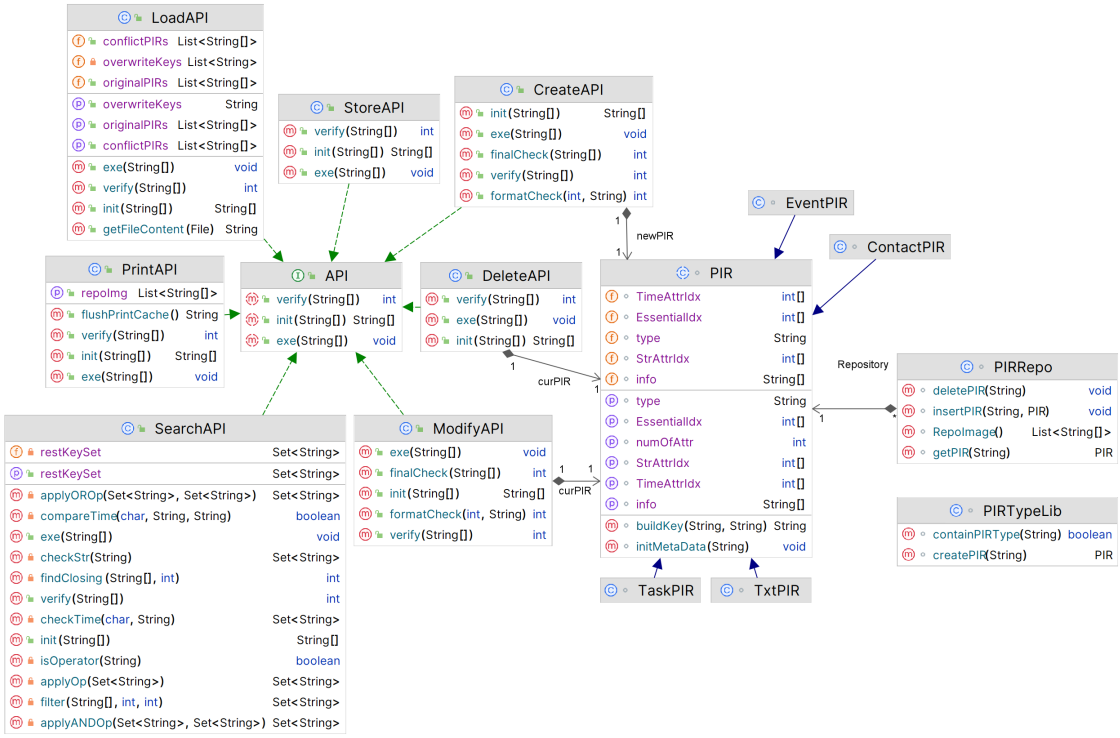

Figure 3: Model

2.1.2 View

The view displays the PIM based on command line view, which is shown in Figure 4.

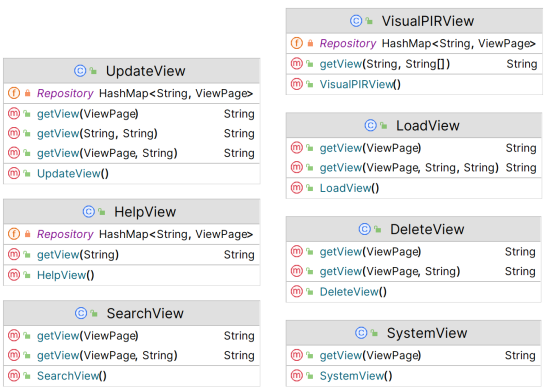

Figure 4: View

2.1.3 Controller

The view handles system input and preliminary processing of data, which is shown in Figure 5.

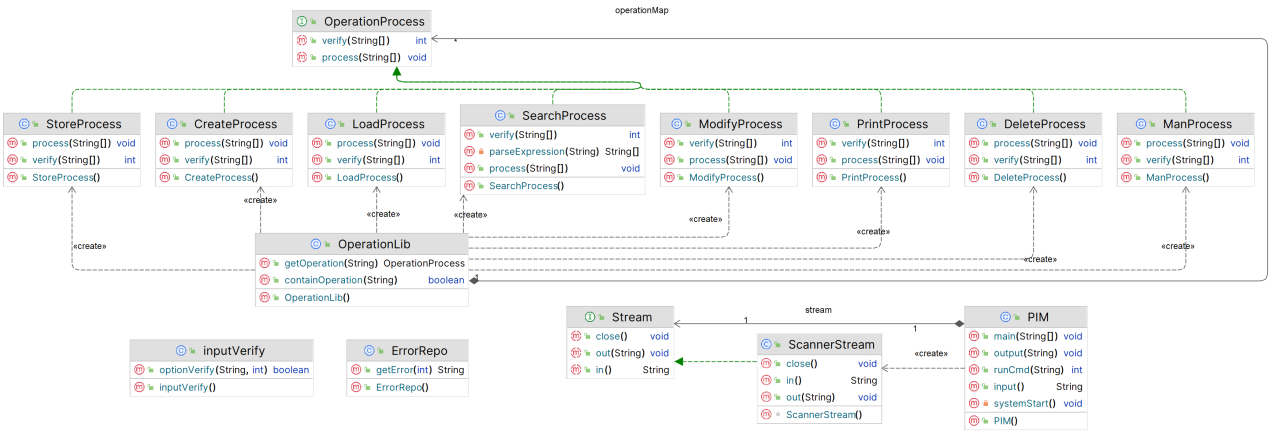

Figure 5: Controller

2.2 Structure of and Relationship among the Major Code Components

2.2.1 Model

- **`API` interface and classes implementing `API`**
  - This interfece defines the behaviors to handle different operations
  - `int verify(String[])`
    - Check data-level errors, such as "PIR type does NOT exists", ets.
    - Return error number else return 0
  - `String[] init(String[])`
    - Initialize essential information to prepare for the execution of operation
    - Return the information back to Controller
  - `void exe(String[])`
    - Execute the operation and save the result in class fields
    - If Controller needs the result, it can call corresponding getxxx() method
  - There may be external fields and methods in the classes implementing `API` interface to handle the operation
- **`PIR` abstract class and classes extending `PIR`**
  - Define essential fields and methods for each kind of PIRs
  - Those classes (`TxtPIR` , `EventPIR` , `TaskPIR` and `ContactPIR`) extending `PIR` only contains the fields to store the PIR data
- **`PIRRepo`**
  - PIR repository which stores all the PIRs with their primary keys
  - Provide methods to insert, delete, get, getall PIRs to/from the repository

- **`PIRTypeLib`**
  - Store all PIR types
  - Provide methods to `new` a specific PIR with a specific type

2.2.2 View

- **Abstract classes**
  - Provide method to access the View Page template and combining filling information with template to generate proper View Page
- **ViewPage**
  - Enum to store View Page template and Stirng format

2.2.3 Controller

- **I/O Stream**
  - `Stream` interface
    - Defines behaviors for I/O stream
- **System Process Controller**
  - `PIM`
    - Parse the command user inputs
    - Organize system process sequance
- **Error handler and Error Repository**
  - `inputVerify`
    - Provide some methods to verify user input format
  - `ErrorRepo`
    - Store all description of errors with their error number
- **Operation Process Controller**
  - `OperationLib`
    - Store all subclasses implementing `OperationProcess` interface with corresponding key (operation type)
  - `OperationProcess` interface
    - Defines the behaviors of process handling operation

2.2.4 Photo of structure and relationship between Model-View-Controller

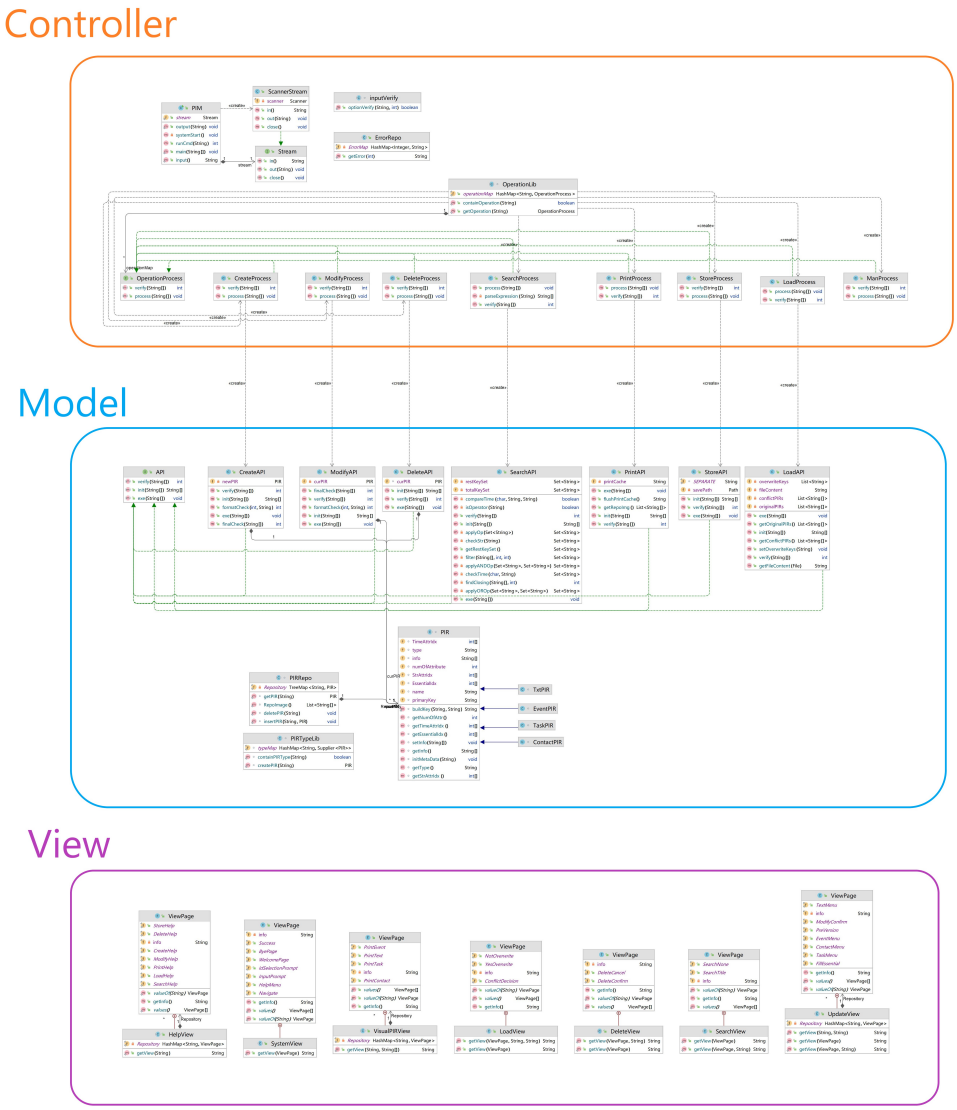

Figure 6: Overview

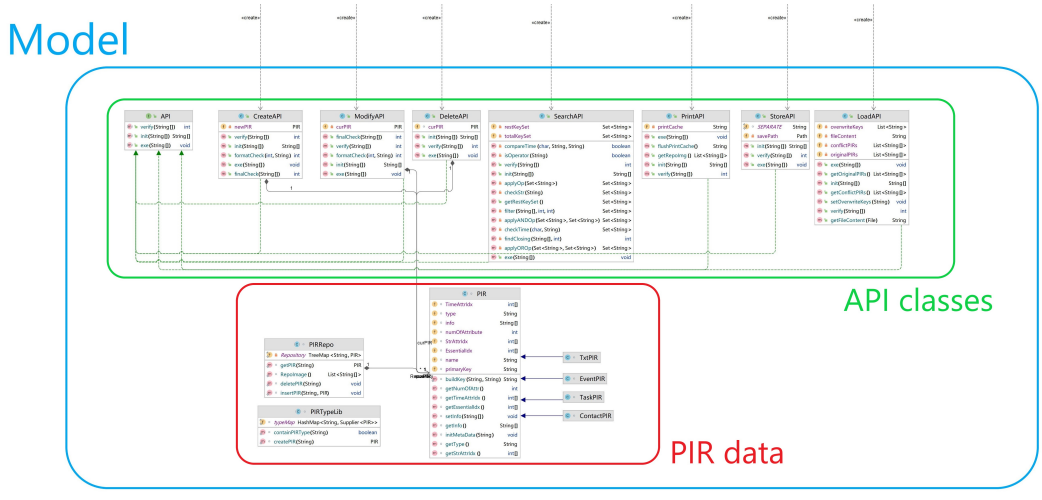

Figure 7: Model

View

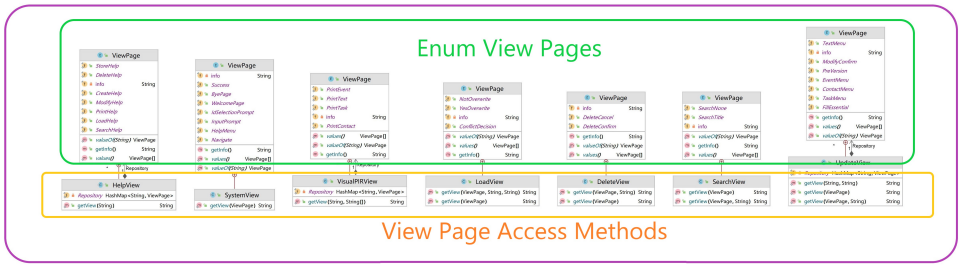

Figure 8: View

Controller

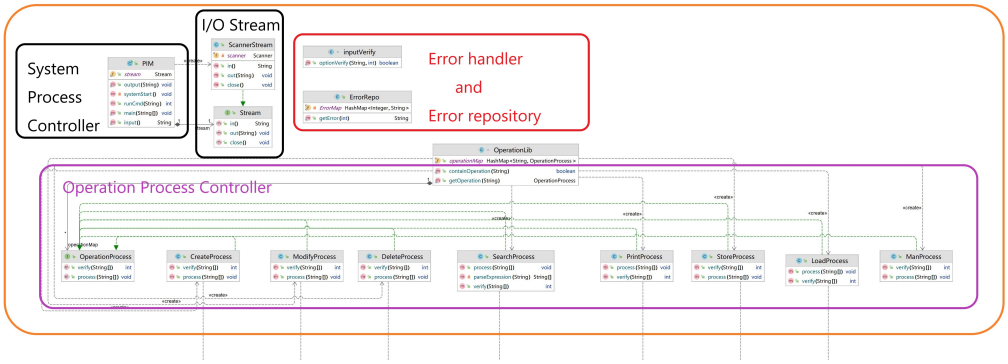

Figure 9: Controller

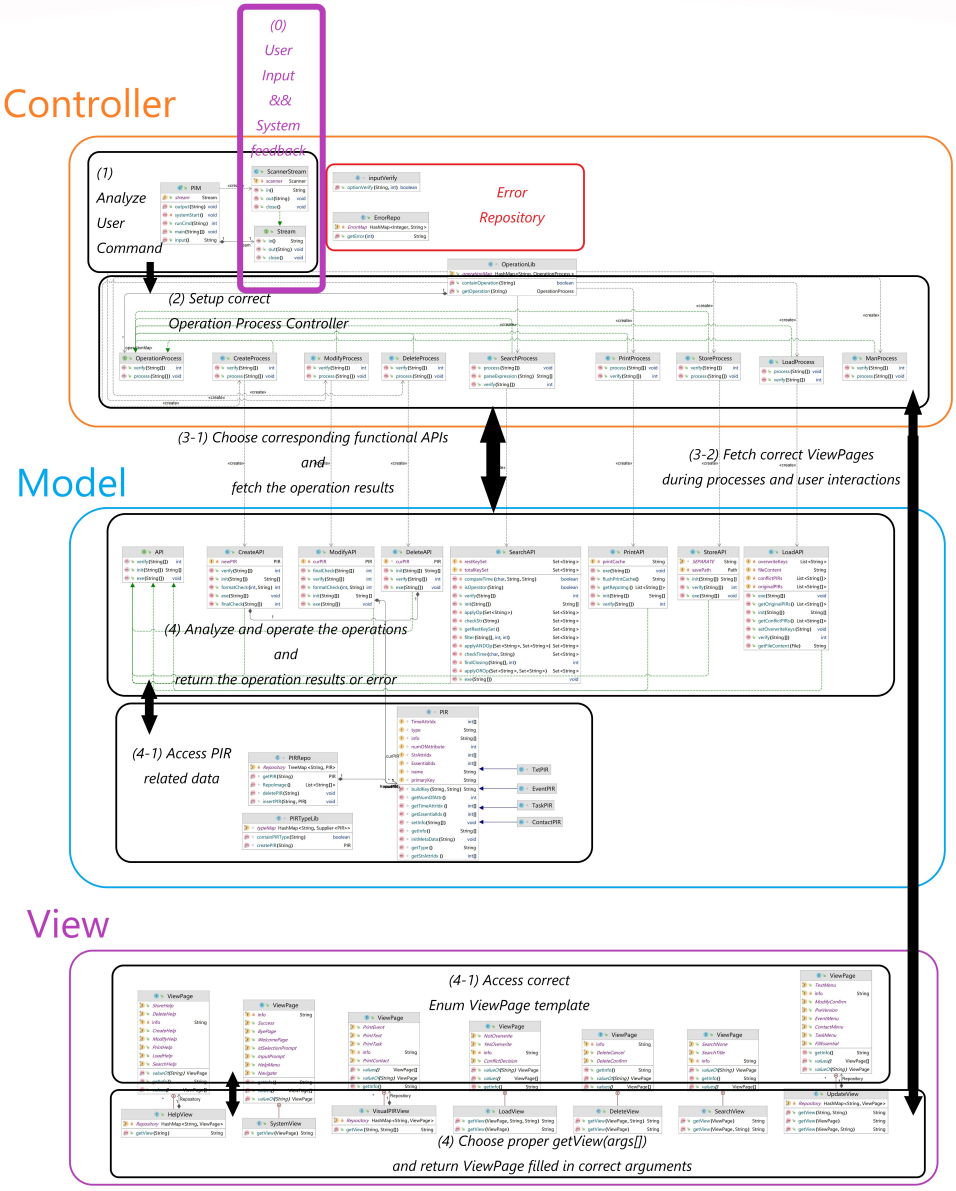

Figure 10: Relationship between Modle-View-Controller

3 Sequence Diagrams

3.1 Diagrams

The Sequence Diagrams of how the PIM system

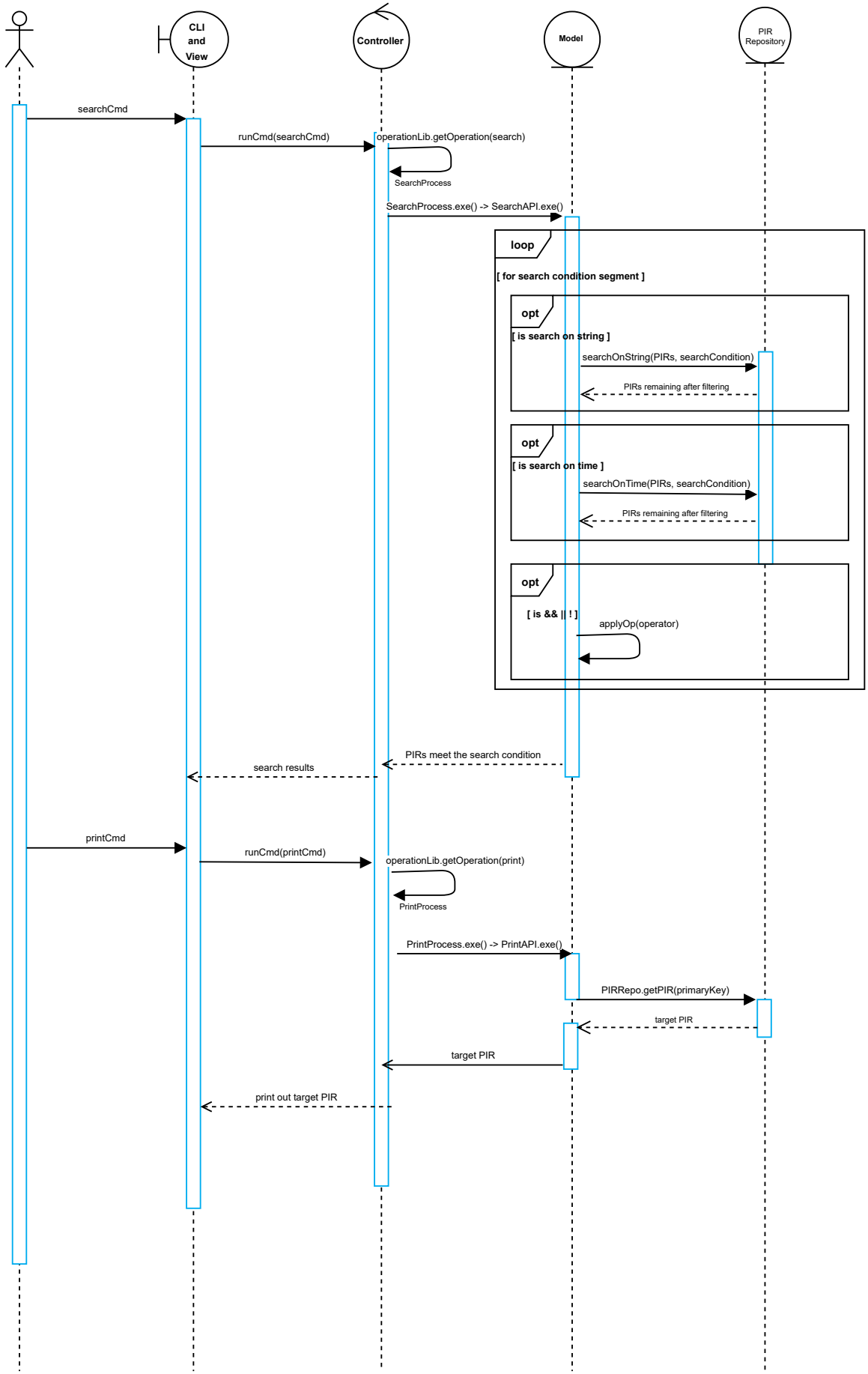

Figure 11: Sequence Diagrams