Model View Controller (MVC)

Motivation

- We use mostly the design patterns together and combine them within the same design solution.

- A **Compound Pattern** combines two or more patterns into a solution that solves a recurring or general problem.

- The **Model-View-Controller Pattern (MVC)** is a compound pattern consisting of the **Observer, Strategy and Composite patterns**.

Model View Controller

- **Model View Controller (MVC)** is a software pattern (paradigm) widely use for implementing user interfaces.

- The internal representation and computation of data is separated from its presentation to the users of the system.

- It decouples the system into 3 main components to achieve many useful features like code reuse and parallel code development.

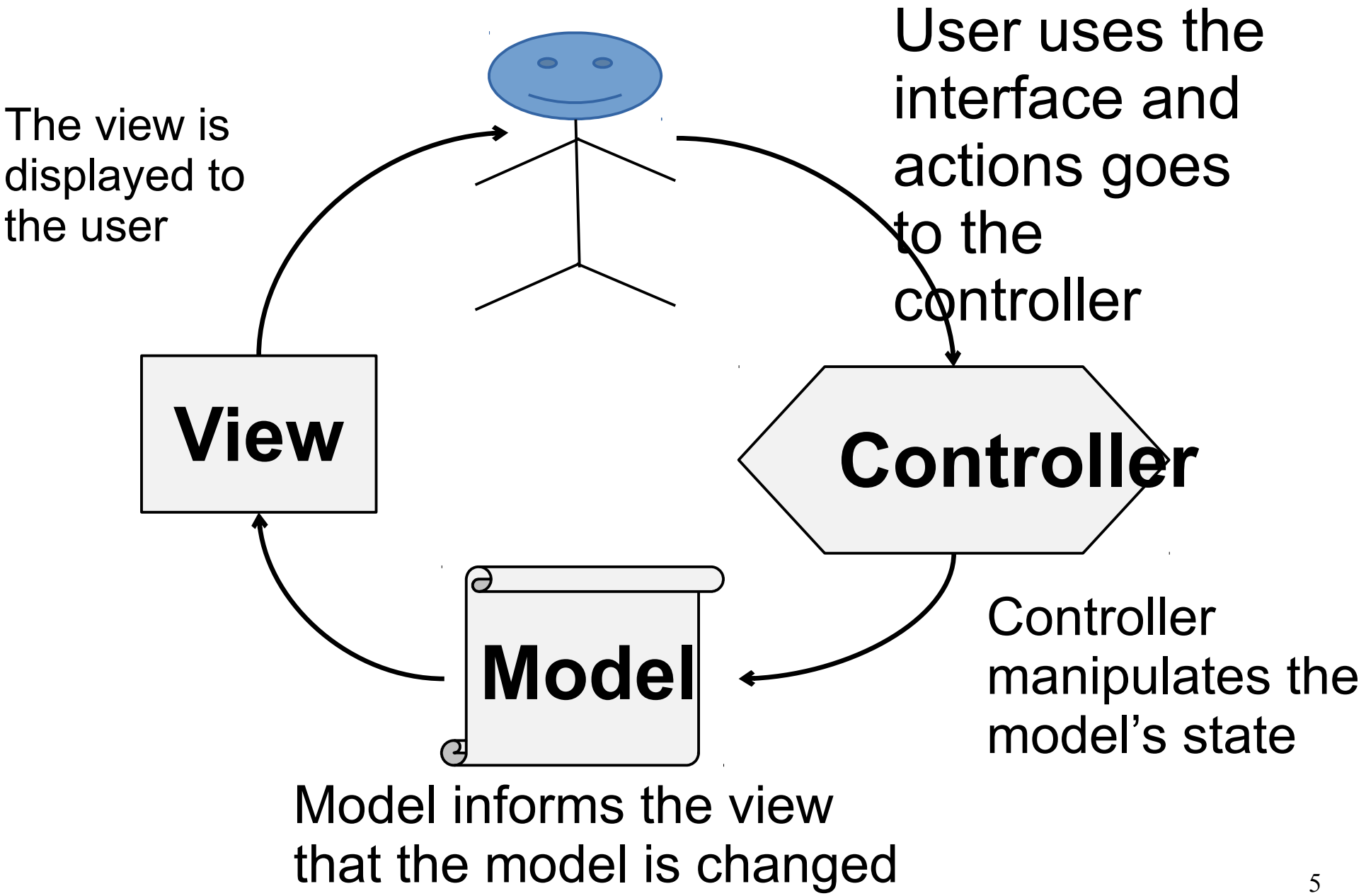
Model View Controller

- **View:** Gives you a presentation of the model. The view usually gets the state and data it needs to display directly from the model.

- **Controller:** Takes user input and figures out what it means to the model.

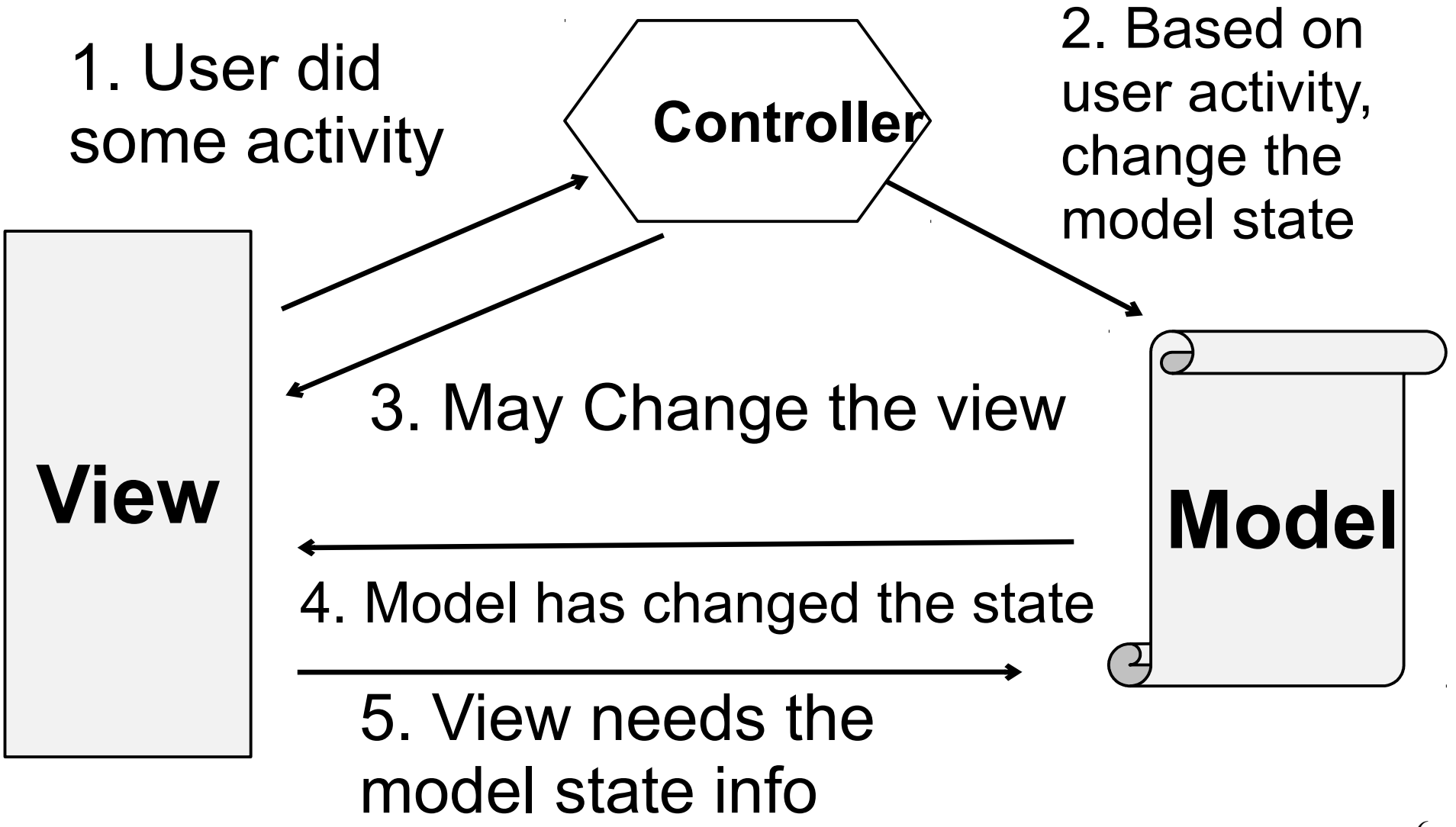
- **Model:** The model holds all the data, state and application logic. The model is oblivious to the view and controller, although it provides an interface to manipulate and retrieve its state and it can send notifications of state changes to observers.

Model View Controller



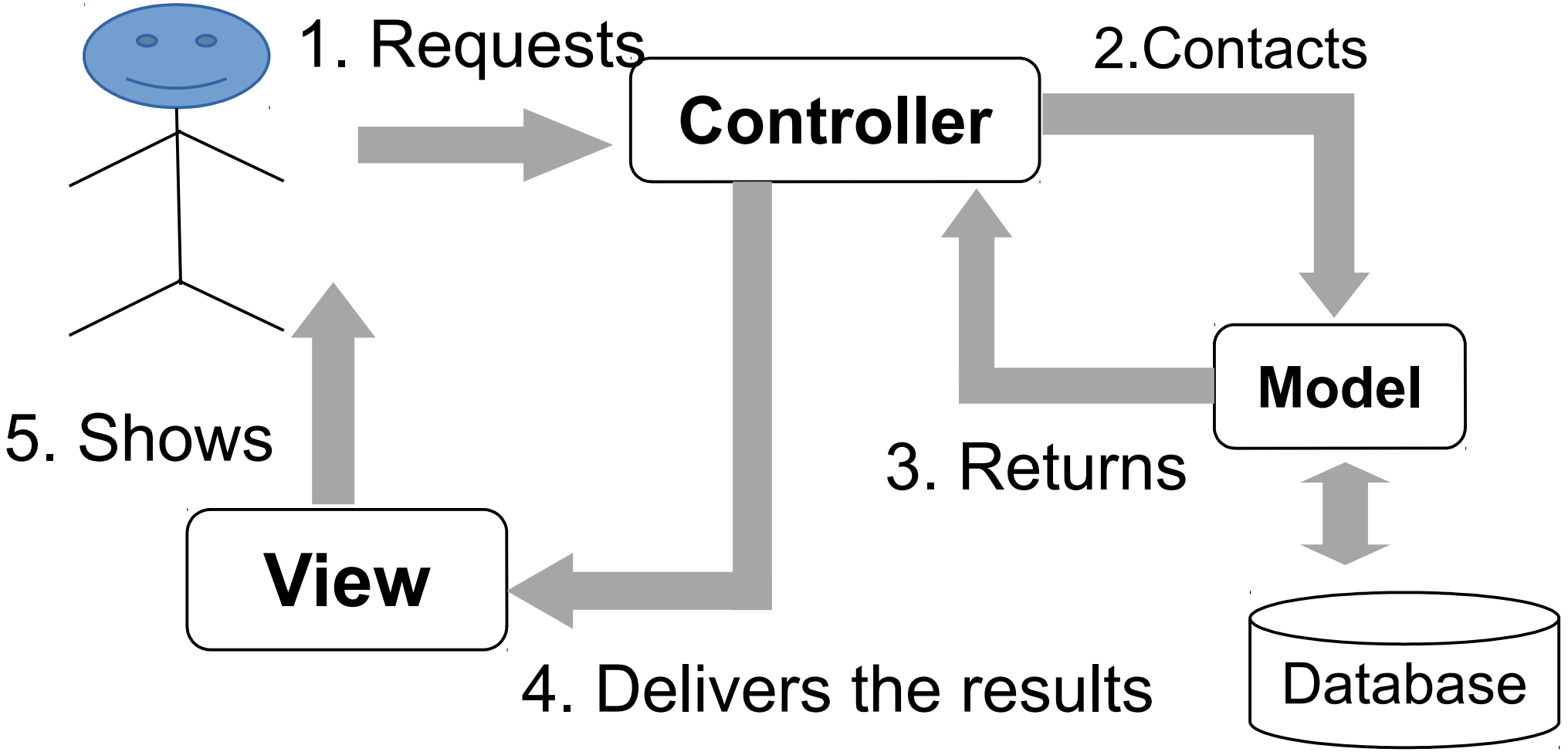
Model View Controller

3 main decoupled components



Overview – Model View Controller

Interactions between Model-View-Controller



Patterns

- **Model:** The model makes use of the **Observer Pattern** so that it can keep observers updated yet stay decoupled from them.
- **Controller:** The controller is the **strategy** for the view. The view can use different implementations of the controller to get different behavior.
- **View:** The view uses the **Composite Pattern** to implement the user interface, which usually consists of nested components like panels, frames and buttons.

- The **Adapter Pattern** can be used to adapt a new model to an existing **view** and **controller**.
- These patterns work together to decouple the three players in the MVC model, which keeps designs clear and flexible.

Strategy Pattern - Controller

- The **view and controller** implement the **classic Strategy** Pattern: the view is an object that is configured with a strategy.
- The **controller** provides the **strategy**. The **view** is concerned only with the visual aspects of the application, and delegates to the **controller** for any decisions about the interface behavior.
- Using the **Strategy Pattern** also keeps the **view** decoupled from the **model** because it is the controller that is responsible for interacting with the model to carry out user requests.
- The **view** knows nothing about how this gets done

Observer Pattern - Model

- The **model** implements the **Observer Pattern** to keep interested objects updated when state changes occur.

- Using the **Observer Pattern** keeps the model completely independent of the views and controllers.

- It allows us to use different **views** with the same **model**, or even use multiple **views** at once.

Composite Pattern – View

- The display consists of a nested set of windows, panels, buttons, text labels and so on.

- Each display component is a composite (like a window) or a leaf (like a button).

- When the **controller** tells the **view** to update, it only has to tell the top view component, and Composite takes care of the rest

Dependencies between Components

- The **Model** knows only about itself.
  - The source code of the Model has no references to either the View or Controller.

- The **View** knows about the **Model**.
  - It will poll the Model about the state, to know what to display. That way, the View can display something that is based on what the Model has done. But the View knows nothing about the Controller.

- The **Controller** knows about both the **Model** and the **View**.

Why dependence hierarchy is used?

- No matter if and how the View class is modified, the Model will still work.

- Even if the system is moved from a systems to other systems, the Model remains the same and can work with no changes.
- In the case of move to other systems, the View probably needs an updated only.

Summary

- We use mostly the design patterns together and combine them within the same design solution.

- One of the widely used design pattern is the Model-View-Controller architecture which is based on a combination of multiple patterns.