

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

Architectural Patterns

Lect. PhD. Arthur Molnar

Babes-Bolyai University

arthur@cs.ubbcluj.ro

- 1 Architectural Patterns
 - Intro
 - Model View Controller
 - Model View Presenter
 - Model View View-Model

- Architectural patterns are higher-level than those previously discussed
- They are about making the code:
 - Scalable, maintainable
 - Enable adding new features quickly
 - Help avoid spaghetti code, or "*crossing the streams*"
 - Easier to testing using mock objects

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- Divide the application into three interconnected parts
- Separates the information from the way it is handled, and the way it is presented
- Popular in using both GUI and web applications

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

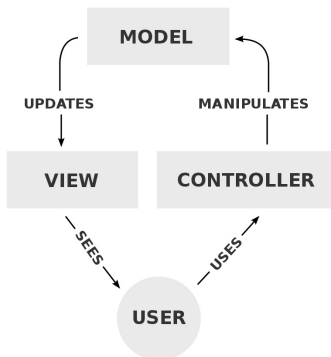


Figure: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

Model

- Central component of the pattern
- It carries the data, manages the logic
- POJO implementation

View

- Representation of the information (GUI, table, chart, etc.)
- The same model can have several different representations (views)

Controller

- Accepts user input, controls the data flow
- Commands both the view and the model

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- The *Model* stores data that is retrieved according to commands from the *Controller* and displayed in the *View*
- The *View* generates new output to the User based on changes in the *Model*.
- The *Controller* can send commands to the Model to update its state (e.g. editing a document).
- *Controller* can also send commands to its associated *View* to change the presentation of the *Model* (e.g. scrolling through a document, movement of document)

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- + Promote simultaneous development
- + High cohesion
 - Enables logical grouping of related controller actions together
 - Views for a given model are also grouped together
- + Low coupling
- + Ease of modification, due to separated responsibilities (easier to enforce the single responsibility principle)

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- Code navigability can be made complex given new abstraction layers and the decomposition criteria of MVC
- Multi-artefact consistency, as feature decomposition can lead to scattering
- Learning curve, as in some cases multiple technologies might be required (e.g. Model 2)

Model View Controller

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

MVC source code

git:/examples/architecture/mvc

MVC source code

git:/examples/architecture/mvc/swing

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

A variation of model-view-controller

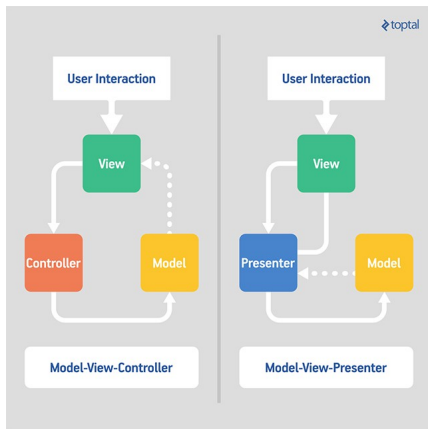


Figure: from <https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c>

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

When compared with MVC

- (Usually) passive View, which renders the UI and routes user input
- The **Presenter** acts as both-ways intermediary between *View* and *Model*
 - Handles user event
 - Retrieves data from models
 - Formats it for display in view
- Each *View* generally has its own *Presenter*
- *Model* is actually an entity that includes business logic and which can update the "proper" model itself

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

A (kind of) sequence diagram

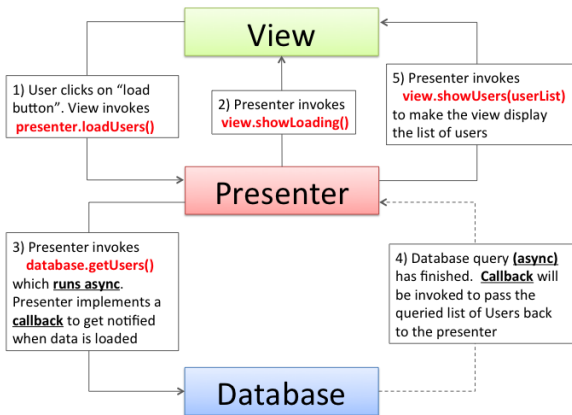


Figure: from <https://github.com/rahulabrol/Messenger>

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

Model

- Holds data that will be used by view
- Collection of classes that represent the business model and process it

View

- Has a reference to *Presenter* and asks it to do work
- Decoupled from the *Model*

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

Presenter

- Coordinates events between the *View* and *Model*
- Responds to view requests, updates the view

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

Testing - MVP *should* make testing easier than MVC

- **View** - test rendering logic and interaction with *Presenter* (which can be mocked)
- **Presenter** - test that the view invokes the correct model method (mock both *Model* and *View*)
- **Model** - test the business logic, mock the data source and *Presenter*

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- + Complex tasks are divided into smaller tasks
- + Less complicated objects, fewer bugs, easier debugging
- + Easier to test
 - Boilerplate code to wire the layers
 - Model cannot be reused and is tied to specific use cases
 - View and Presenter are tied to data objects since they share the same type of object with the Model

Model View Presenter

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

Example

git:/examples/architecture/mvp

Model View View-Model

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

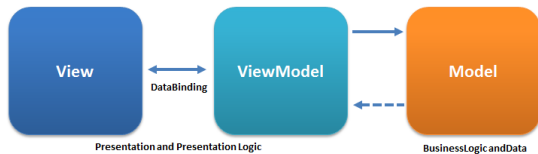


Figure: from <https://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93ViewModel>

- Separate development of the GUI (using source code or a markup language - XAML) from its back-end and business logic
- Invented by Microsoft architects to simplify GUI event-driven development
- Developed to take advantage of the data-binding mechanisms in WPF by removing GUI-code from the view itself

Model View View-Model

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- Allows late-breaking changes to the user interface (component layout, dimensions, presentations) without affecting application code
- The *View* is the only component that is platform-specific
- The *ViewModel* acts as a value converter between data model objects and GUI components
- ViewModel can act as a mediator between GUI components and its back-end

Model View View-Model

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural
Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

The **Model**

- Can be the domain model itself, or the data access layer that represents content
- No references should appear to either *View* or *ViewModel*

The **View**

- Platform-specific code for the user interface
- May not reference the *Model*
- Bidirectional binding to *ViewModel*
 - Communication to ViewModel is represented by Commands from the View to the ViewModel
 - Communication from ViewModel is represented via the data binding and the update of the data from the ViewModel

Model View View-Model

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

The **ViewModel**

- Abstraction of the *View* exposing public properties and commands to facilitate communication between *View* and *Model*
- References the Model, might reference the View
- Abstraction of View's code-behind, reusable code when modifying the *View*

Model View View-Model

Lecture 05

Lect. PhD.
Arthur Molnar

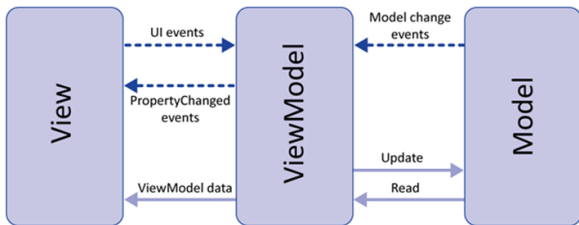
Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model



- **View - ViewModel:** communication is called Data Binding (bidirectional)
- **ViewModel - Model:** communication is done via Notifications (bidirectional)

Model View View-Model

Lecture 05

Lect. PhD.
Arthur Molnar

Architectural Patterns

Intro

Model View
Controller

Model View
Presenter

Model View
View-Model

- + Reduce the amount of code-behind and dependency between it and view-specific code
- + Model does not change to support a View
- + Separate designers from coders by separating GUI code from code-behind, reducing development time
 - Create more files
 - Simple tasks can be more complicated
 - Lack of standardization, most specific to Windows Presentation Foundation (WPF)