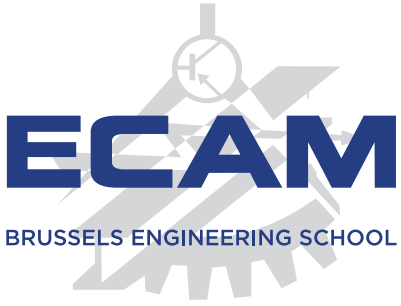


I402A Software Architecture
and Quality Assessment

Session 7

Interaction-Oriented Architecture

Sébastien Combéfis

Fall 2019

Objectives

- Interaction-oriented architecture description
  - Separation between interaction, data and business logic
  - The *Model-View-Controllller* (MVC) pattern
  - The *Presentation-Abstraction-Control* (PAC) pattern
- Model-View-* variations of MVC pattern

  MVC, MVVM and MVP
- Choice criteria between different existing patterns

  According to use-case, quality requirement and chosen technology

Interaction-Oriented Architecture

- First objective to separate interaction/data
  - Separation of data abstraction and business processing
  - Typically used for interactive applications
- System split into three main partitions
  - **Data** abstraction and business logic
  - **Control** flow and system configuration actions
  - Data **view** and interface for user input

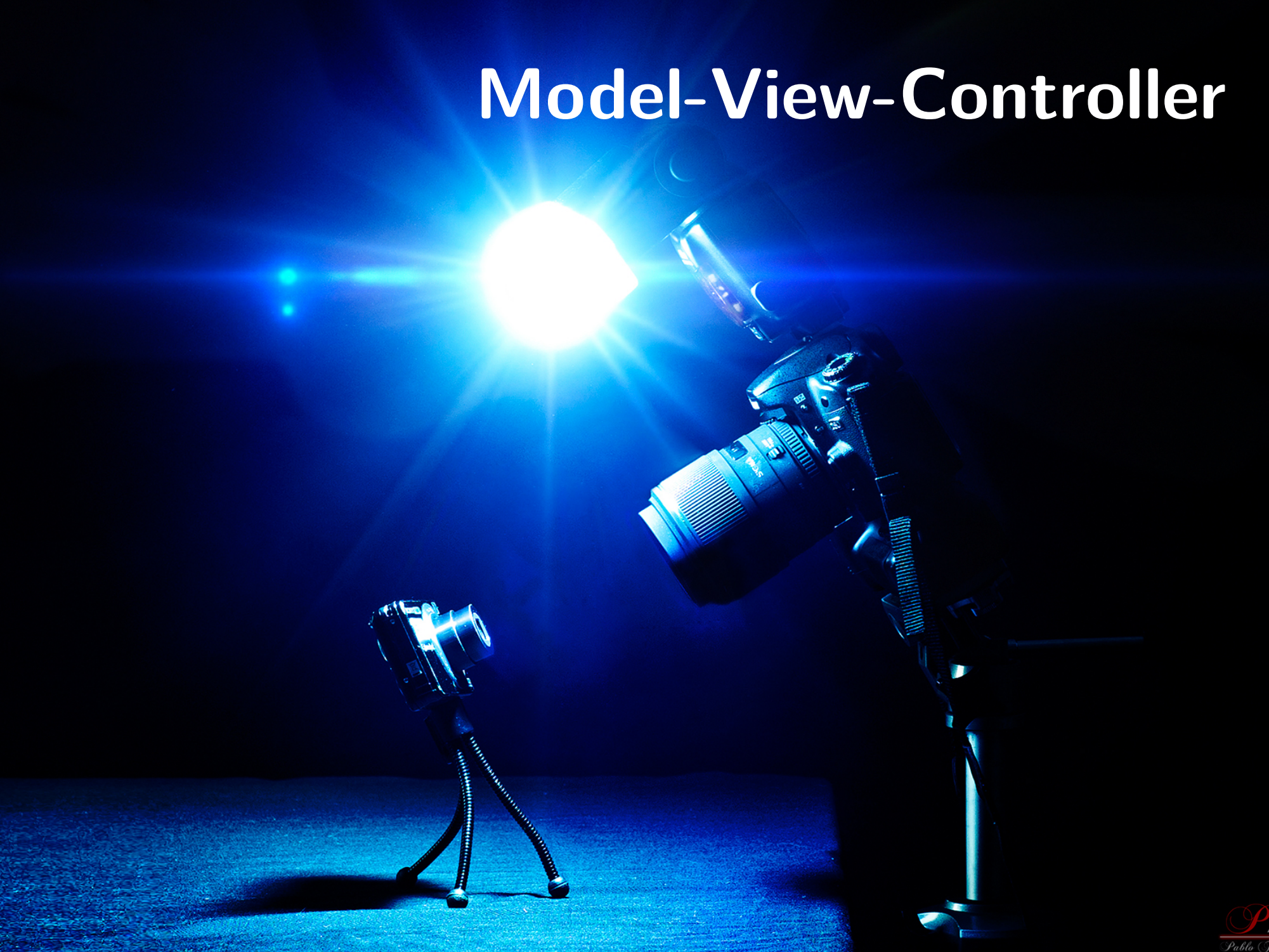
Main Models

- Two main models that are interaction-oriented
  - Model-View-Controller and Presentation-Abstraction-Control
  - Decomposition into three components
- Need for interactions via an user interface

  Web/mobile/desktop application, physical device, etc.
- Differences in control flow and organisation

  PAC is hierarchical and agent-based and MVC is unclear

Model-View-Controller



Model-View-Controller (1)

- Separation of the view for the user of internal data

  View of the presented and accepted information from the user

- Decomposition into three interconnected parts
  - The **model** encapsulates data and business logic

    What the application is
  - The **controller** reacts to actions and orchestrates execution

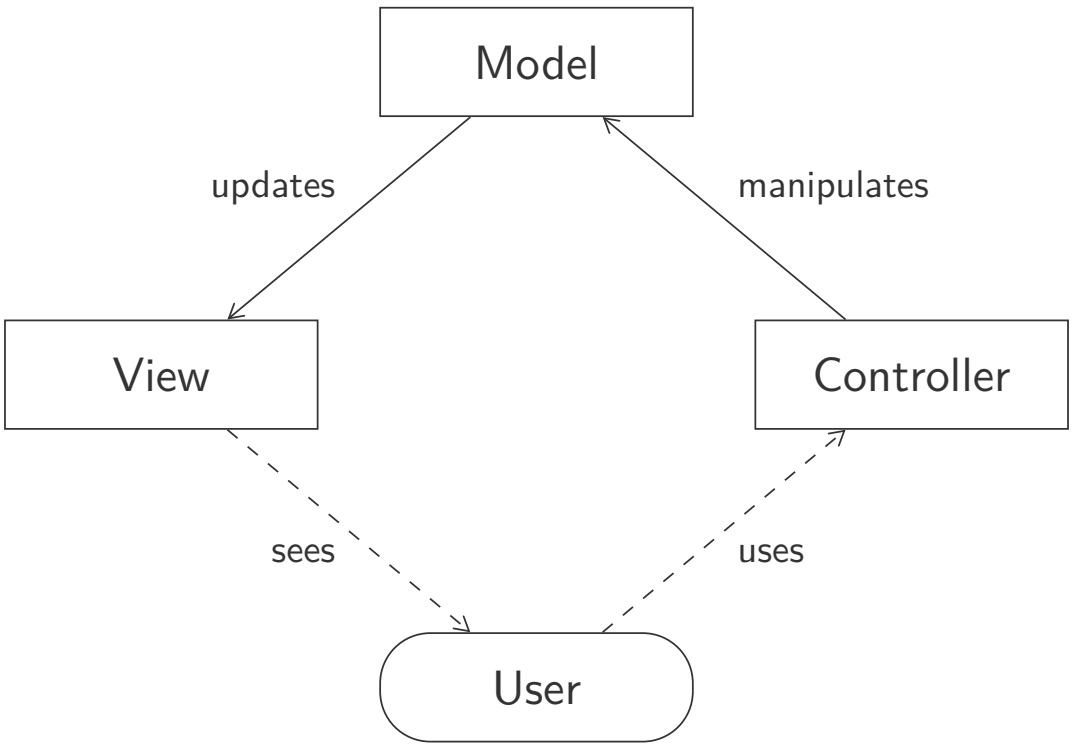
    How the model is presented to the user (logical UI)
  - The **view** presents the model data to the user

    The "minions" of the controller

Model-View-Controller (2)

- The model is isolated from the view and controller

  It is created first and the others are registering to it



Cook-Customer-Waiter

- Cook takes order from waiter and prepare meal for customer

  Represents the data and the logic of the app

- Customer places order and receives food through waiter

  Only part of the app with which user acts directly

- Waiter takes request from customer to cook

  Interprets users inputs to connect model and views

Model

- Central component that manages data

  Know the logic and the constraints on the data

- Stores the raw data of the application and the logic
  - Capture the behaviour of the application domain
  - Central structure of the application
- Notification of the state changes
  - To the view to change the presentation of data
  - To the controller to change the available commands

View

- Representation of data in various forms

  Diagram, table, text, etc.

- Several presentation components
  - Request information to the model to generate a representation
  - Possibility to have several views for the same data

Controller

- Accept the inputs from the user to handle them

  Transmit commands to the model or the view

- Several processing components
  - Interface between models and associated views and inputs
  - Commands dispatch to the model for updating...
  - ...and to associated views to change the presentation

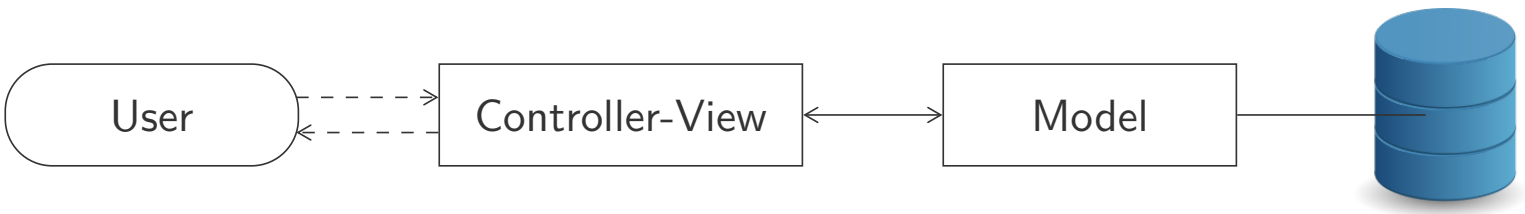
MVC-I

- Simplified version of MVC with two components

  Controller and view merged for input/output and processing

- Connection between model and controller-view

  With an observer pattern, for example

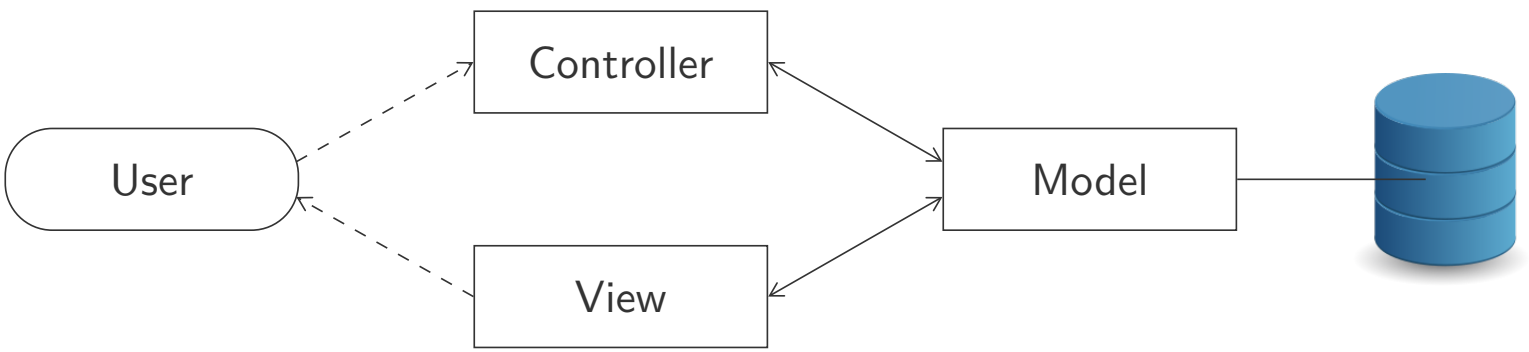


MVC-II

- Improvement of MVC-I with three components

  Separation between view and controller modules

- Model and view initialisation and connection by the controller
  - The view displays data
  - The controller accepts requests and validates data
  - The model has the same role as in MVC-I



MVC Application

- MVC adapted for interactive application with several views

  Clear division between modules eases team work

- **Advantages**
  - Several views synchronised on the same data model
  - Easy to add/modify views
- **Disadvantages**
  - Can make expensive a modification of the data model
  - Division between view and controller not always clear

A Pattern View

- The model uses the Observer pattern

  Notification of views and controllers of state changes

- The view and the controller implement the Strategy pattern

  Controller is the behaviour of the view and can be exchanged

- The view uses the Composite pattern internally

  Management of window buttons and other components

ECAM TV Example (1)

https://github.com/ECAM-Brussels/ECAMTV

ECAM TV Example (2)

- Several widgets whose content may change

  Widgets are also clickable to have a full view

- The application is a page composed of several widgets

  Must be able to run in parallel on multiple TV screens

- Uses external services as a source of information

  Number of calls to those services must be minimised

Presentation-Abstraction-Control

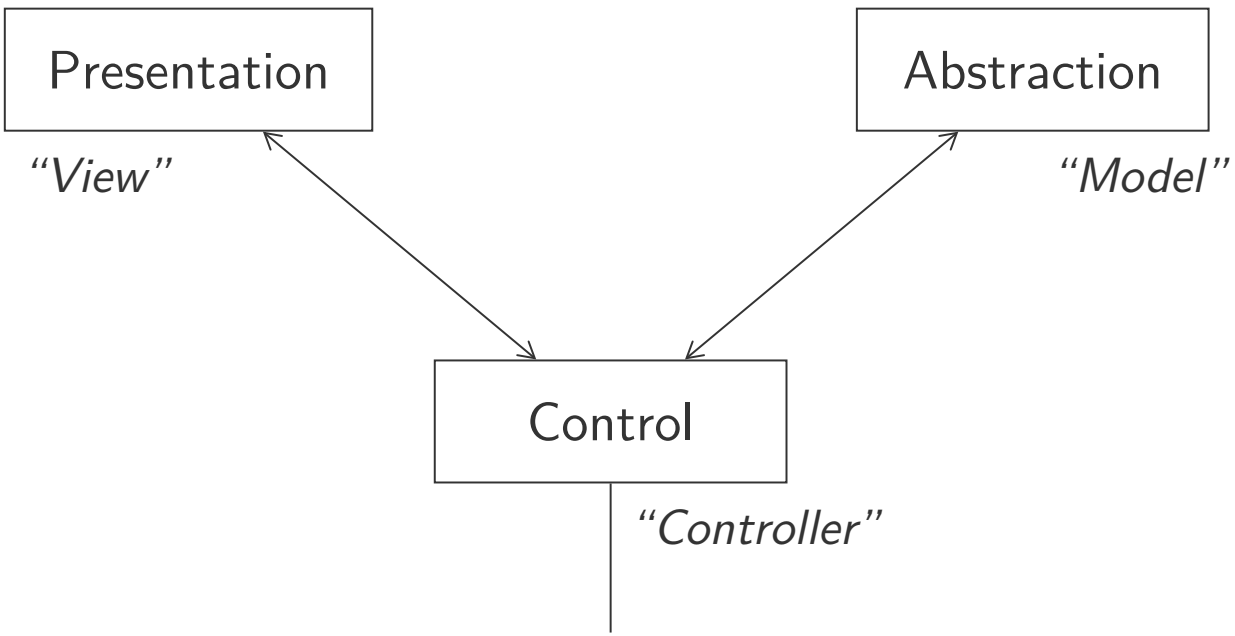
Presentation-Abstraction-Control (1)

- System decomposed into several cooperative agents

  Developed from MVC for applications with agents

- Each agent consists of three components
  - The **presentation** visually formats data
  - The **abstraction** collects and processes data
  - The **control** manages the control flow and the communication

Presentation-Abstraction-Control (2)



PAC Agent

- A PAC agent is very similar to an MVC application

  Presentation/View, Abstraction/Model, Control/Controller

- Control makes the communication presentation $\leftrightarrow$ abstraction

  No direct link view $\leftrightarrow$ model as with MVC

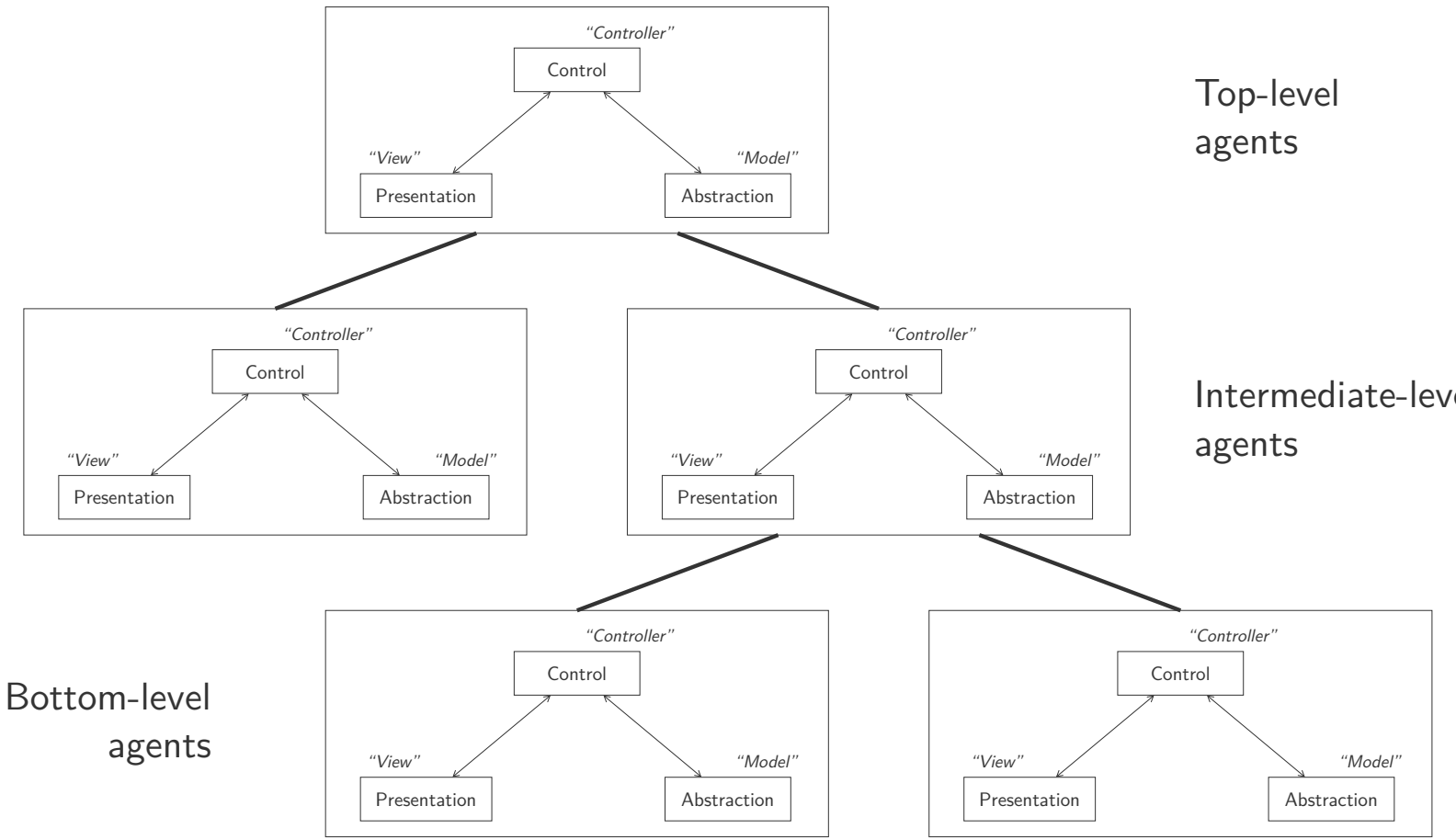
- A software system consists of several PAC agents

  Possibilities for interaction between the different agents

Multiple Agent (1)

- Agents hierarchy organised in levels
  - The high level agent contains data and core logic
  - Bottom agents have specific data and presentation
  - Intermediate agents coordinate the lower ones
- Several properties on different agents
  - Each agent has its specific job (party isolation)
  - Presentation not necessarily present for intermediates
  - Control mandatory for inter-agents communication

Multiple Agent (2)



Top-level agents

Intermediate-level agents

Bottom-level agents

PAC Application

- Several hierarchical cooperative agents weakly coupled

  Suitable for distributed systems, interfaces with rich widgets

- **Advantages**
  - Supports multi-views and multi-tasking (concurrency)
  - Reusing agents, extension by adding new agents
- **Disadvantages**
  - Overload and heaviness caused by communications
  - Difficult to identify the agents and their number

Model-View-*

Model-View-*

- Original MVC developed with Smalltalk'80 by Krasner & Pope

  In order to integrate an UI with the application domain

- Emergence of several variants from the 80s

  Subtle but very important differences

- Three major families of MV* patterns
  - Model-View-Controller (MVC)
  - Model-View-View-Model (MVVM)
  - Model-View-Presenter (MVP)

MVC/PAC Evolution

- Birth of MVC with the emergence of GUI in the 80s

  Then arrival of PAC in 1988 once MVC documented

- Many evolutions to MVC towards the end of the 90s
  - Technologies: touchscreen, voice input, web/mobile app, etc.
  - Languages: event, generic type, lambda expression, etc.

- Same basic principle of separation of concerns

  Model of the application domain/user interface

Inter-Component Communication

- All the MV* patterns have the model and view components

  *The * defines the communication between M and V and with user*

- Consistency between state and view by synchronisation
  - By **flow** with sequential execution of commands

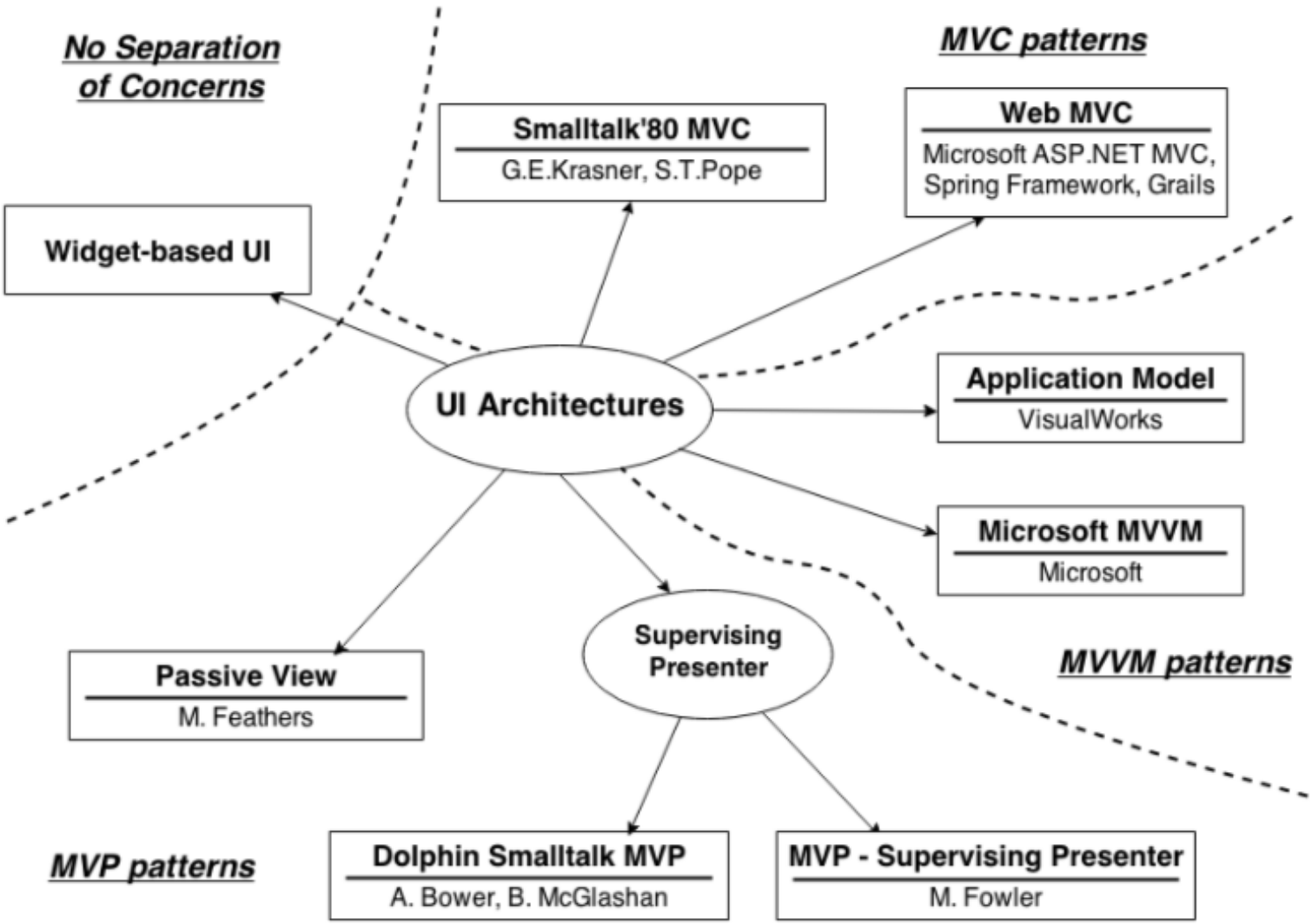
    Direct call between interface and domain components
  - By **observer** with notifications system

    Domain notifies observers that have registered

- Flow for small applications otherwise not maintainable

  Observer more complex for domain change

World of MV* Models



Widget-Based UI

- Deposit and arrangement of widgets on a window

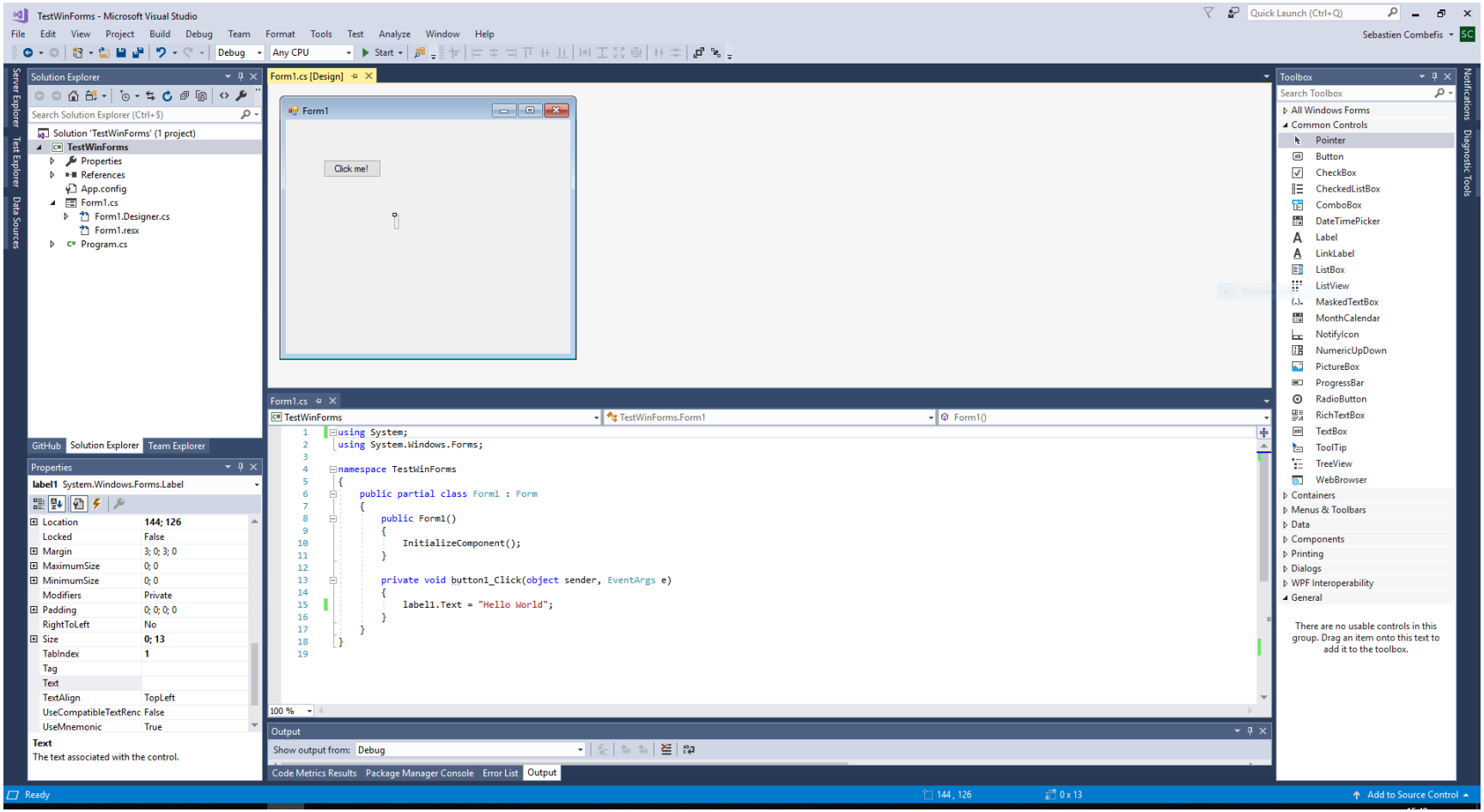
  Logic management code in a `Form` class

- Complete access to the interface widgets and to data

  No separation of concerns

- **Advantages**
  - Simplicity to understand with widgets as instance variables
  - Consistency thanks to a flow synchronisation
  - Development efficiency for small applications

Windows Forms



Model-View-Controller (MVC) Family

- MVC initially developed for desktop application

  With a very rich graphical user interface

- Evolution of the pattern with technology and new needs

  Used today for web/mobile application

- Two main representatives of the MVC family
  - The original **Smalltalk'80 MVC**
  - The **Web MVC** variant for web applications

Smalltalk'80 MVC (1)

- Three modules for the Smalltalk'80 MVC

  Store and manage data, display them and manage user input

- Model completely isolated from the other components

  Application logic does not depend on presentation

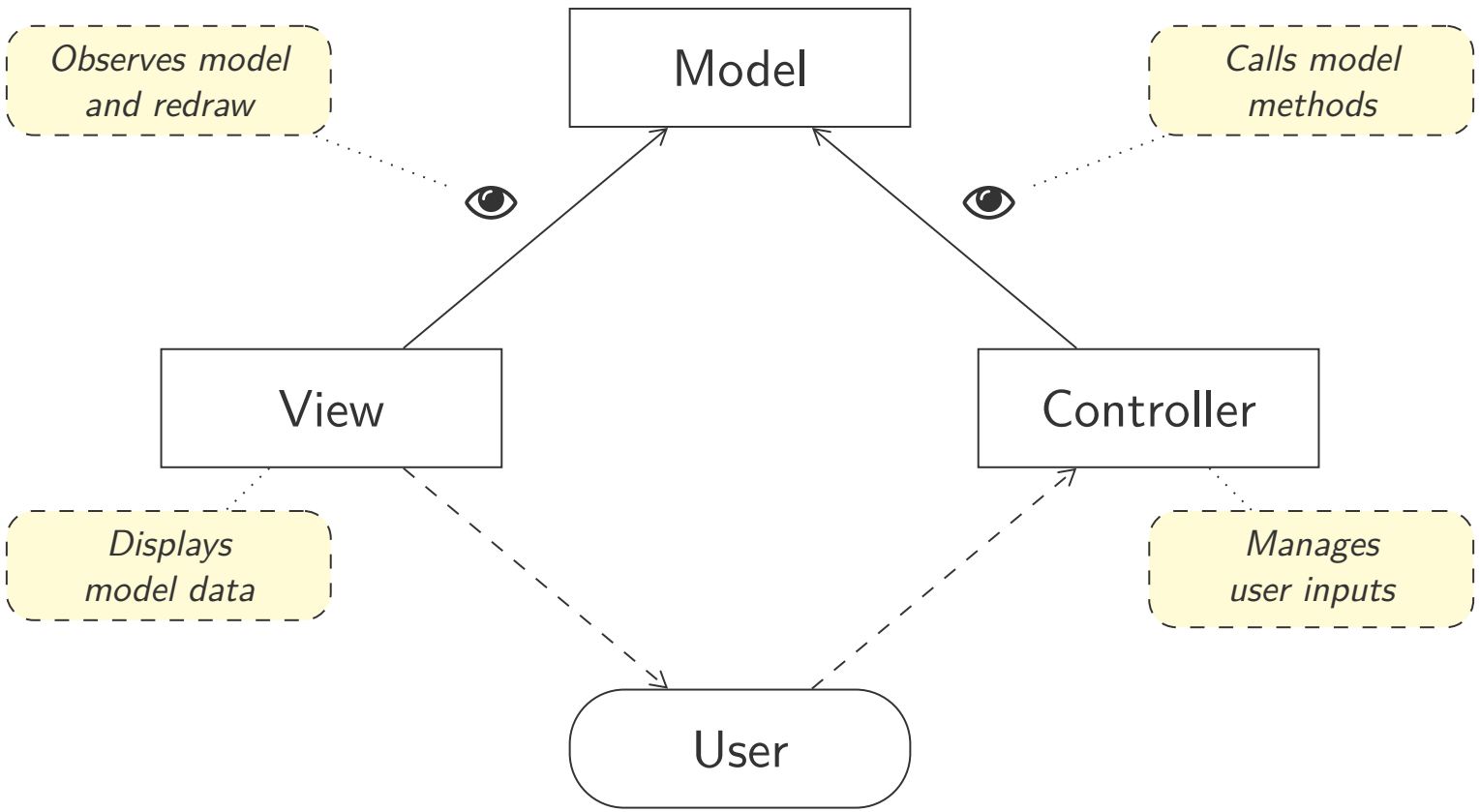
- Interaction pair view/controller for the user

  Synchronisation by observer with the model

- Weakness for some interface requests

  Is "text in red if negative" a property of the view or model?

Smalltalk'80 MVC (2)



Web MVC (1)

- Web MVC for web development on the server side

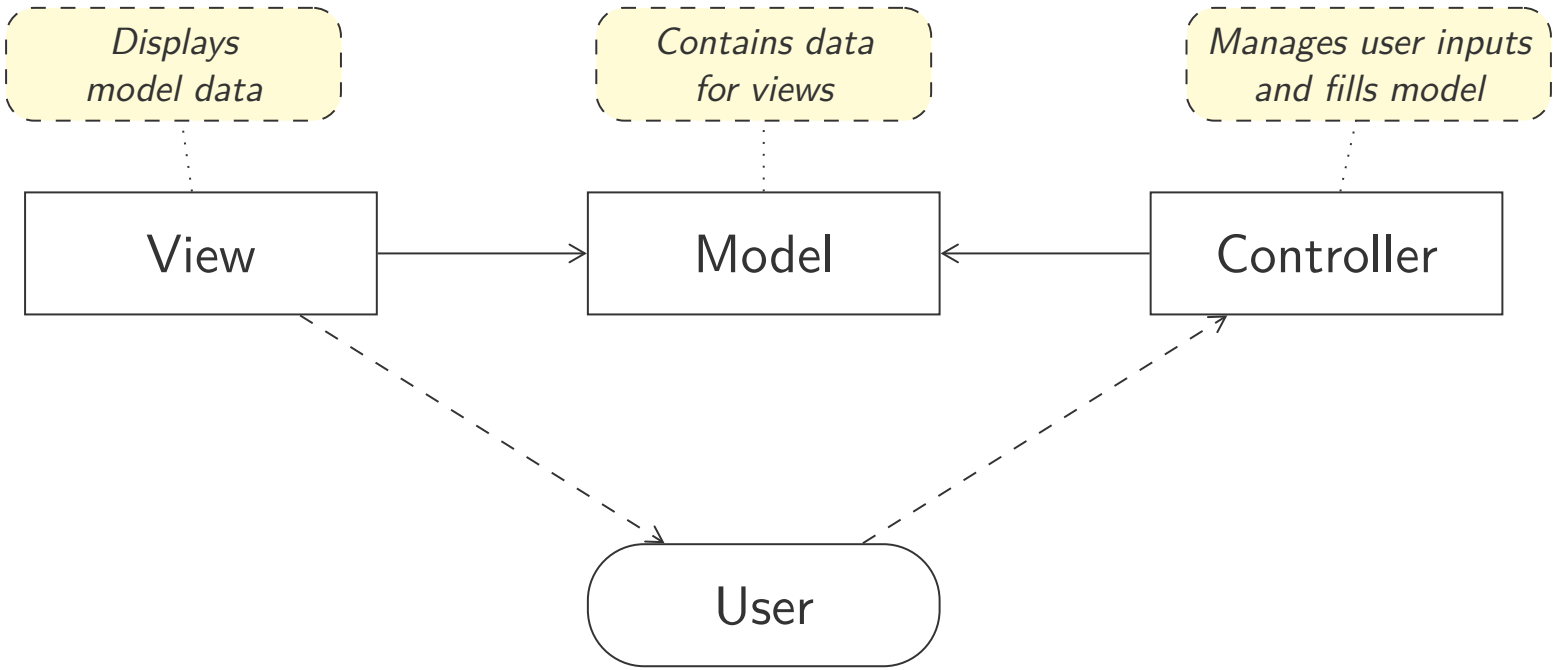
  For example ASP.NET MVC, Spring, Grail, etc.

- Three modules with the view being the HTML content

  Model stores data and controller manages user action

- Responsibility differences with the Smalltalk'80 MVC
  - Application logic triggered by the controller
  - The model just stores data to generate views
  - The controller is responsible for creating the model

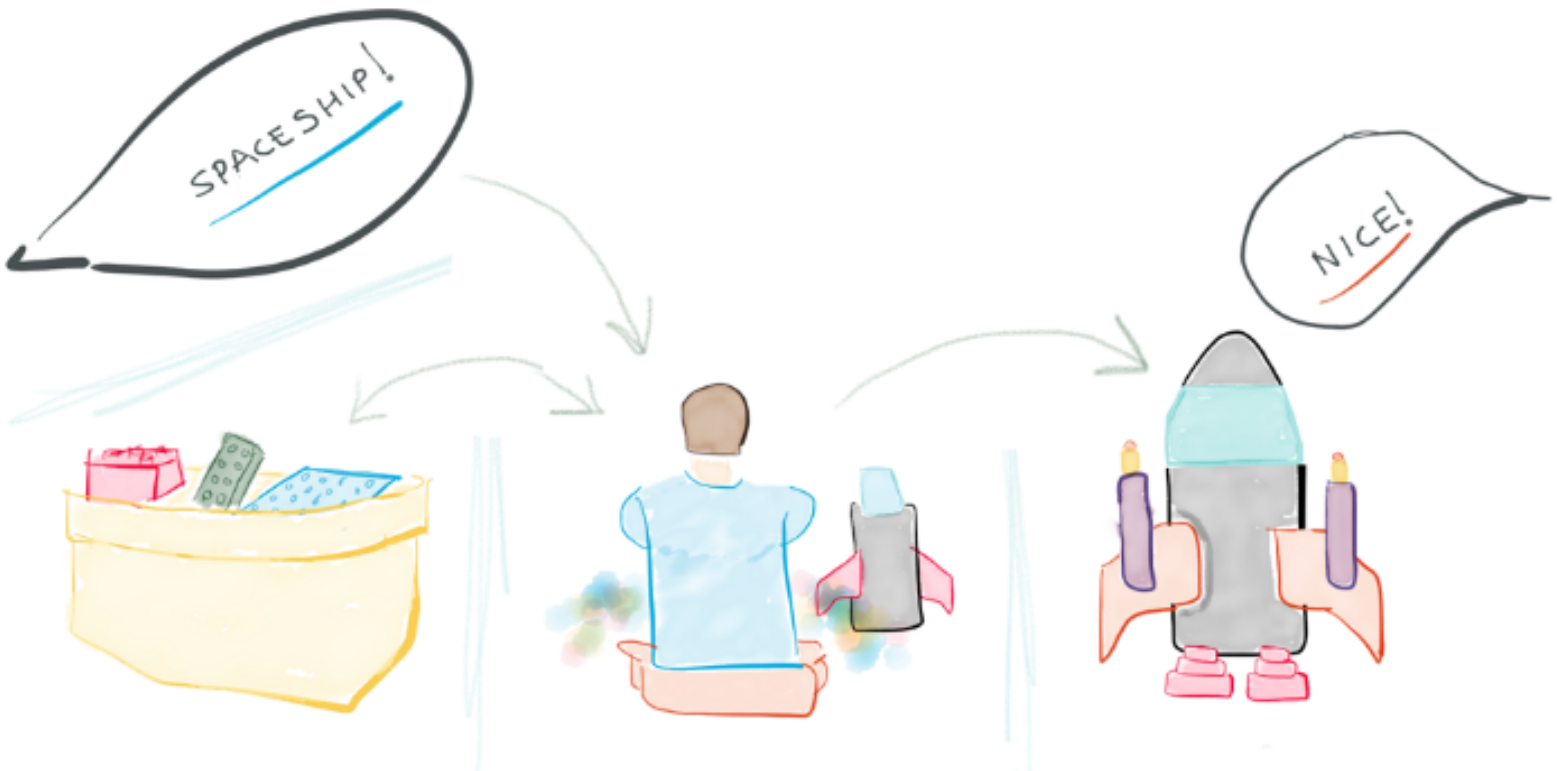
Web MVC (2)


Displays model data
Contains data for views
Manages user inputs and fills model
View
Model
Controller
User

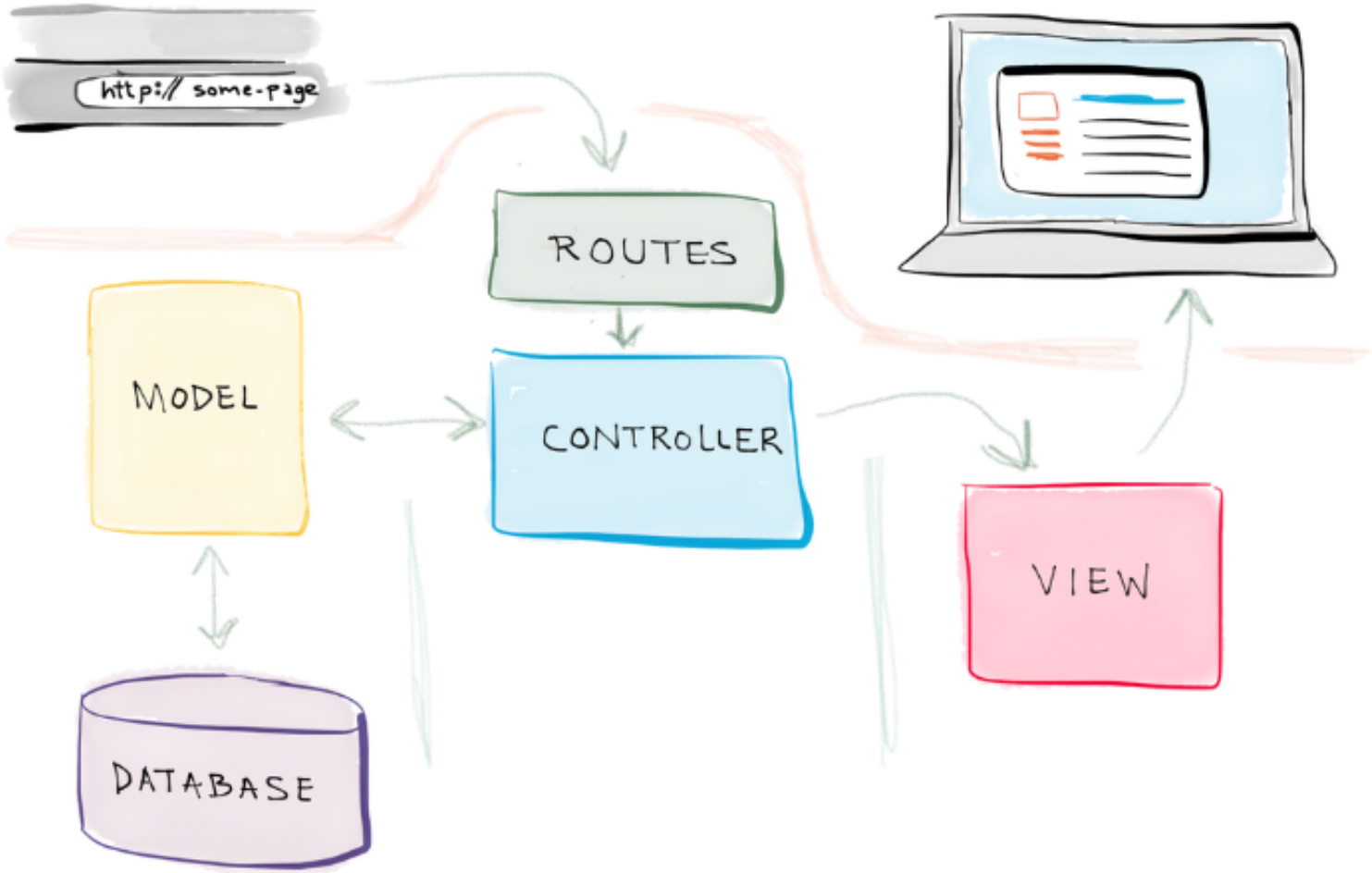
MVC Explained with Legos (1)

- You brother (user) asks you to build a spaceship

  With bricks (model), you (controller) build the spaceship (view)



MVC Explained with Legos (2)



Model-View-View-Model (MVVM) Family

- Supporting "view state" since not in the model

  For example, to be able to display in red the negative numbers

- Two components to the model
  - **Domain model** for the state of the application domain
  - **Presentation model** for the view state

- Merging the Model-View pair from MVC

  And the view interacts with the presentation model

Application Model (1)

- Application Model in VisualWorks implem. of Smalltalk

  Data storage/manag., display, user input and view state

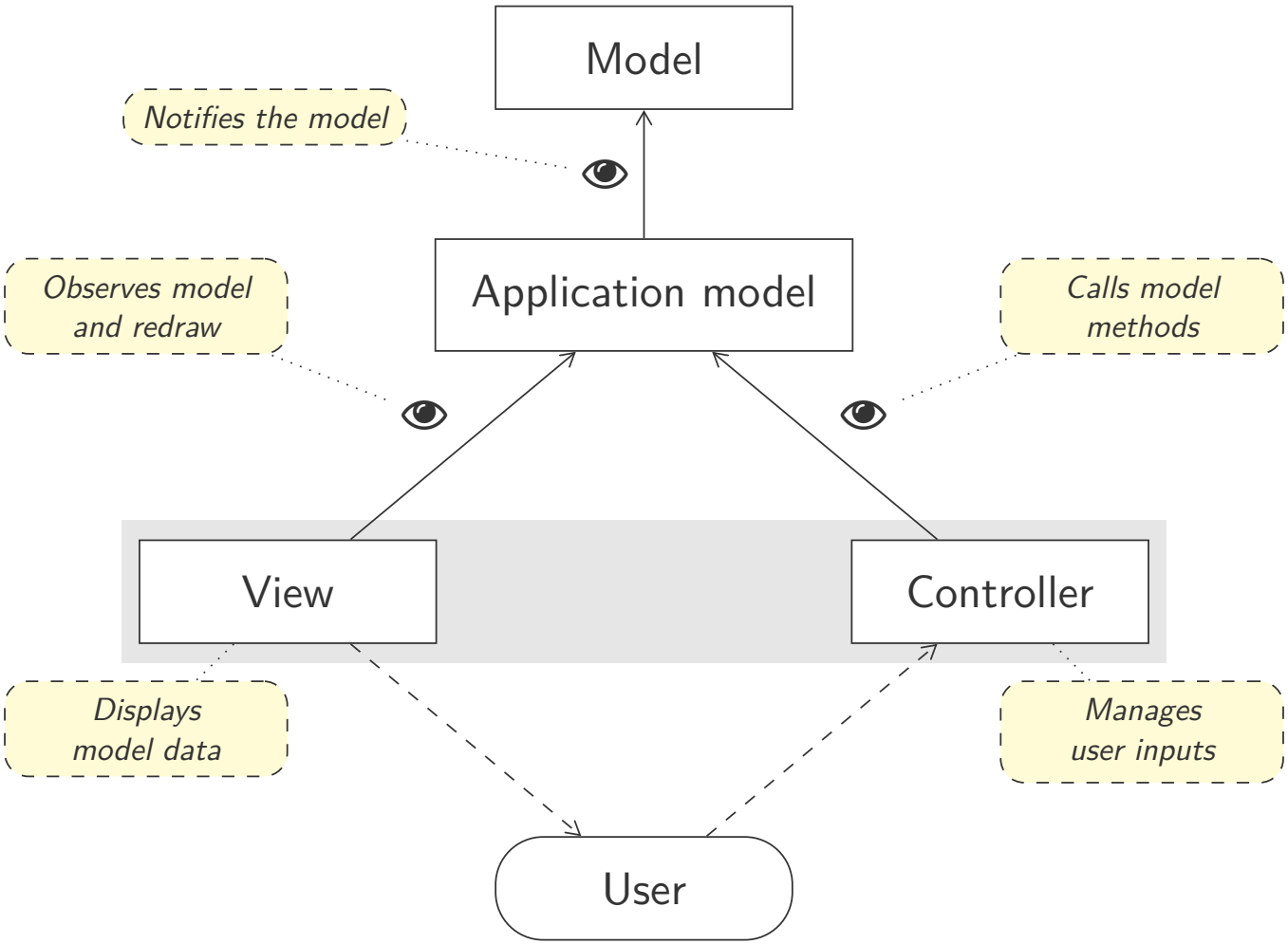
- View-Controller indirectly communicates with the model

  Processing of user inputs before sending to the model

- Requires additional development time

  Creation of widgets and custom adapters

Application Model (2)



Microsoft MVVM (1)

- Three components including the model for domain data

  View-Model for view state and user input, and View

- Declarative link between View and View-Model

  No additional logic in the View

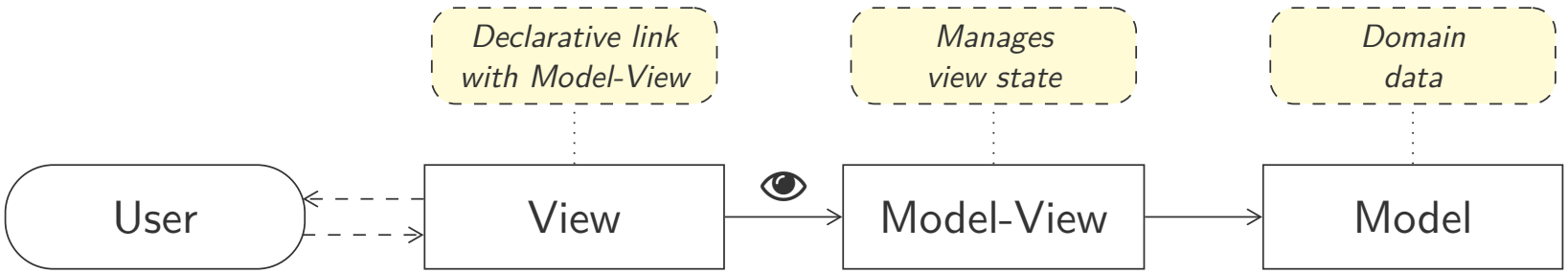
- Same data in several forms and simultaneously

  Multiple Views for one View-Model, several by Model

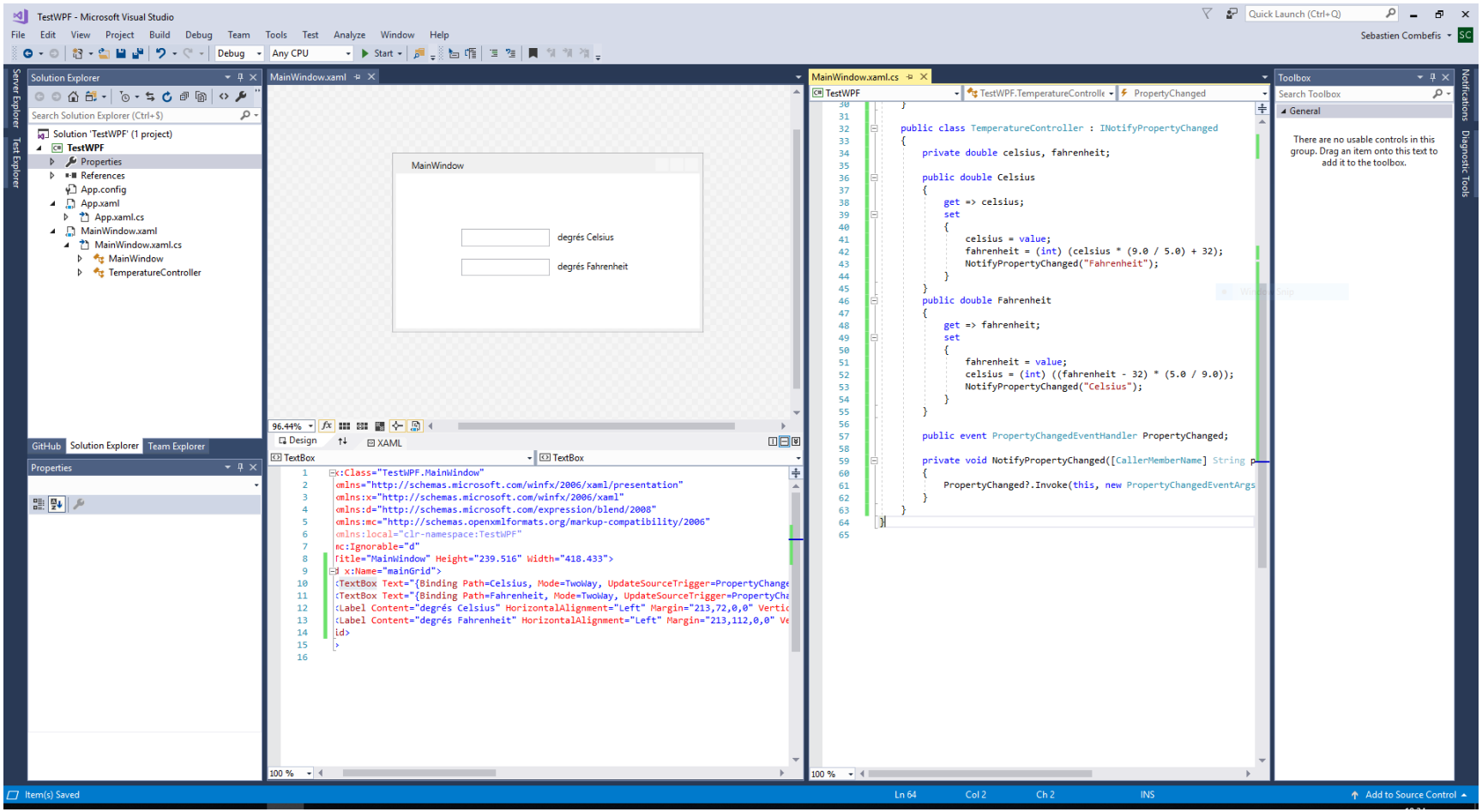
- Declarative data binding between View and View-Model

  Link between properties of a widget and View-Model data

Microsoft MVVM (2)



Microsoft WPF



Model-View-Presenter (MVP) Family

- Introduced by IBM mid 90s

  Popularised by the Dolphin Smalltalk pattern

- Presenter component which supervises the view

  Also manages user events and can change the view

- Two major trends according to synchronisation
  - **Observation**: Dolphin Smalltalk and Supervising Presenter
  - **Flow**: Passive View

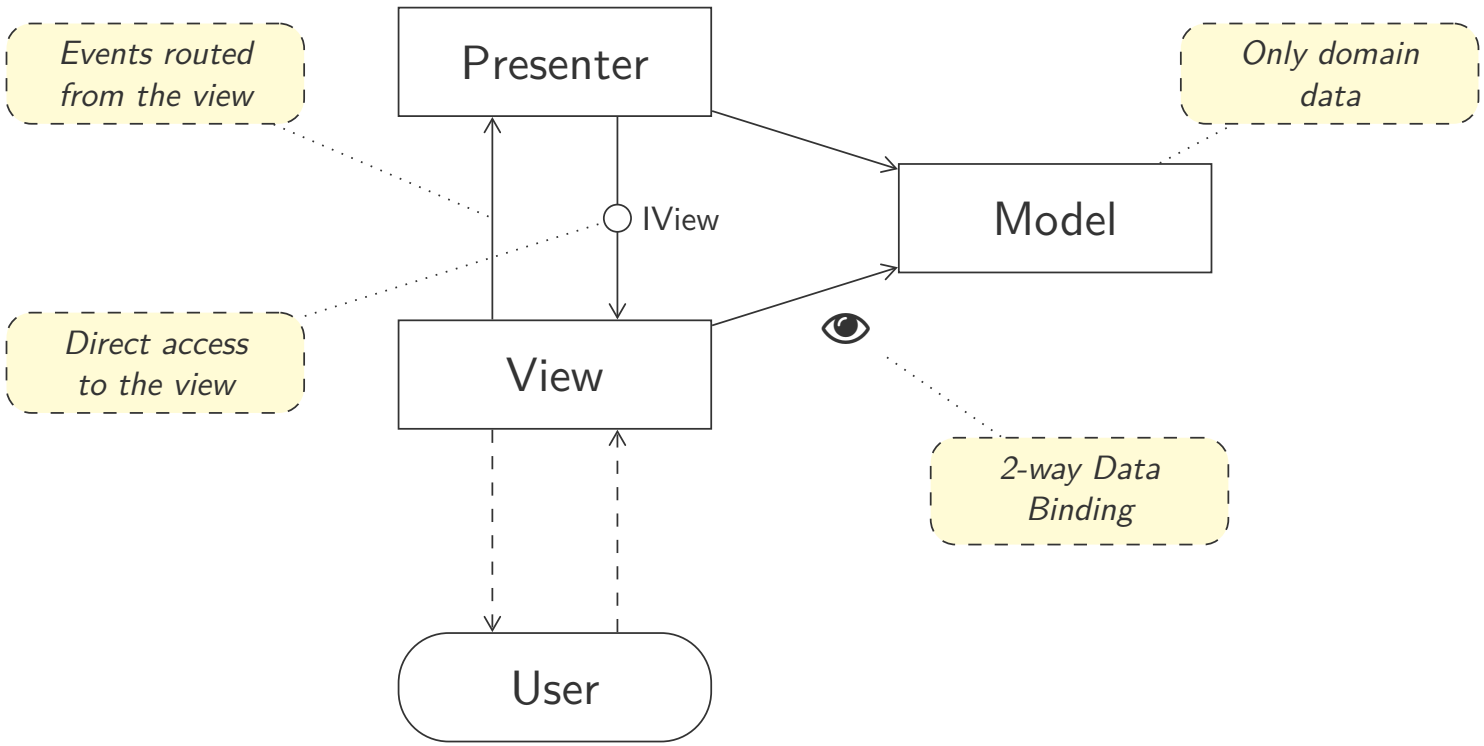
Dolphin Smalltalk MVP (1)

- Adaptation of responsibilities and roles compared to MVC
  - The **model** manages domain data
  - Basic user inputs and interface rendering by the **view**
  - Supervision of View-Model synchronisation by the **presenter**
- Direct access to the model and view for the presenter

  Application core moved from model to presenter
- Delegation of user actions to the presenter

  The View(-Controller) only does basic processing

Dolphin Smalltalk MVP (2)



Supervising Presenter (1)

- Domain data and presentation state in the model

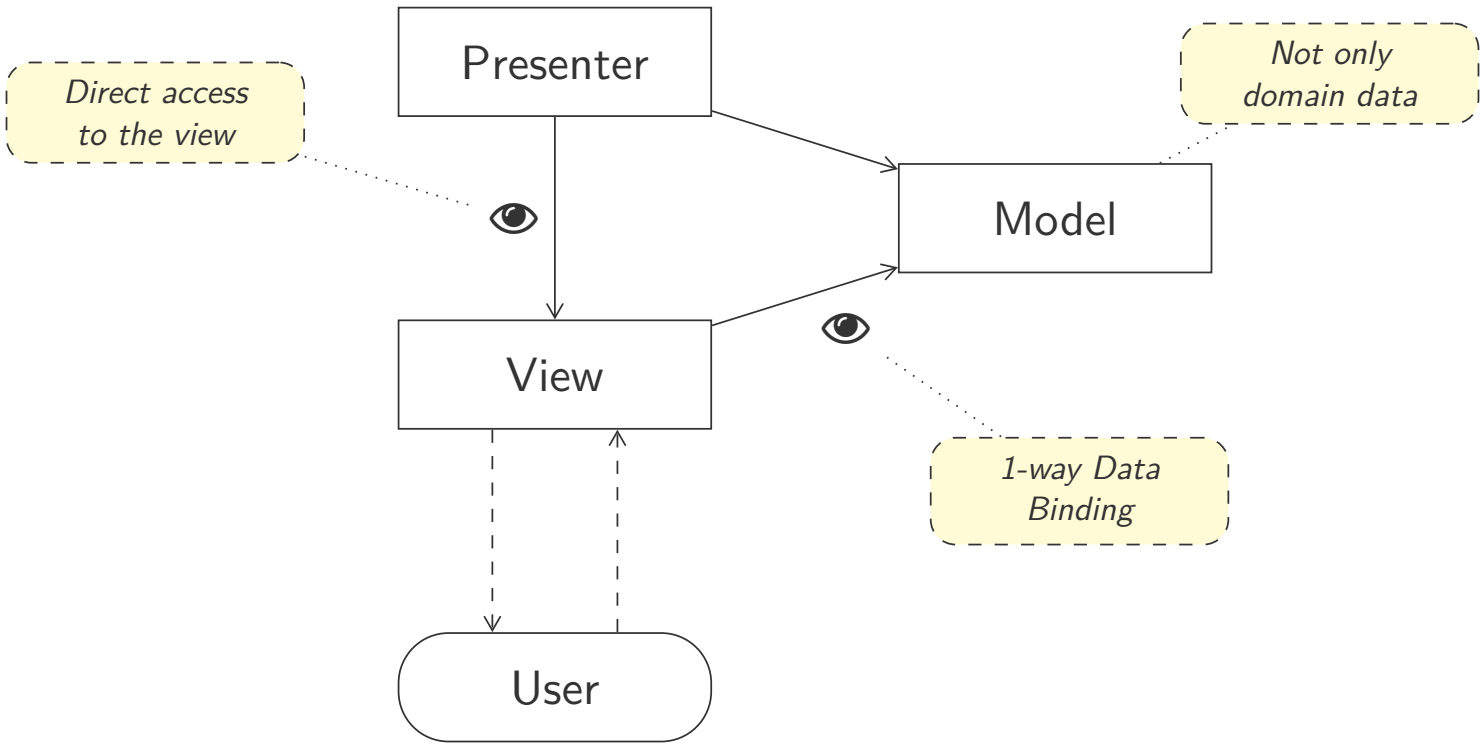
  View maps UI and data, presenter handles complex view logic

- Reconciliation to the Widget-Based UI model

  Through the direct interaction between presenter and view

- Presenter plays a coordination role between view and model
  - Manages user input, updates model/view, calls domain logic
  - Observes the view to relay modifications on the model
  - Knows the view well, difficult to have several

Supervising Presenter (2)



Passive View (1)

- Data domain in the model and representation by the view

  Synchronisation of the user and view state by the presenter

- Similar to the Microsoft MVVM model

  But synchronisation by flow rather than by observer

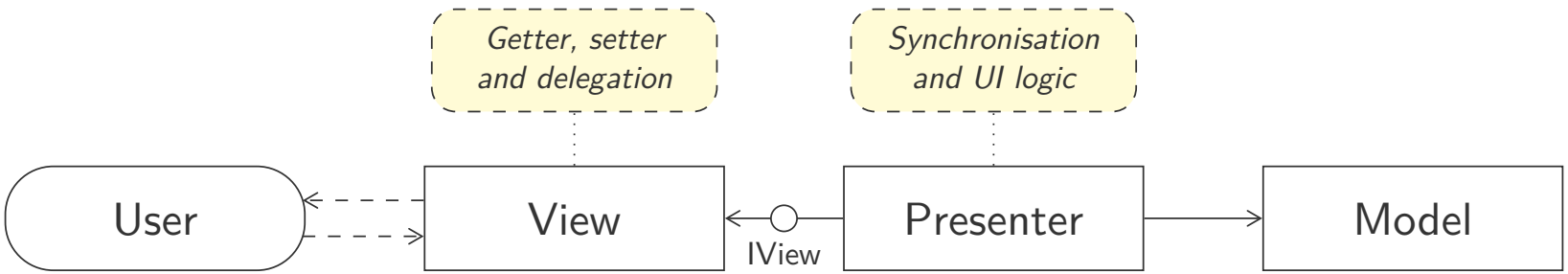
- The view is simple and minimal, in particular simplifying tests

  Getter, setter and logic to delegate events

- Code difficult to maintain if very rich and complex UI

  Same flaw as with Widget-Based UI

Passive View (2)



References

- Joseph Spinelli, *MVC Overview*, December 19, 2018.
  https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5
- Mohaideen Jamil, *Model 1 and Model 2 (MVC) Architecture*, January 14, 2014.
  https://www.simplecodestuffs.com/model-1-and-model-2-mvc-architecture
- Tidalwave, *Beyond MVC: better UI design with PAC, Presentation Model and DCI*, September 6, 2012.
  http://tidalwave.it/fabrizio/blog/beyond-mvc-pac-presentation-model-dci
- Artem Syromiatnikov and Danny Weyns (2014). A Journey Through the Land of Model-View-* Design Patterns. In *Proceedings of 2014 IEEE/IFIP Conference on Software Architecture*. IEEE.
- Alex Coleman, Real Python, *Model-View-Controller (MVC) Explained ? With Legos*, December 20, 2014.
  https://realpython.com/the-model-view-controller-mvc-paradigm-summarized-with-legos
- Ankit Sinhal, *MVC, MVP and MVVM Design Pattern*, January 3, 2017.
  https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad
- Bipin Pandey, *Model View Presenter(MVP) in Android with a simple demo project*, December 6, 2017.
  https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42

Credits

- Pablo Fernández, July 20, 2010, https://www.flickr.com/photos/hadock/4975562025.
- Carniolus, May 23, 2017, https://en.wikipedia.org/wiki/File:Minions_characters.png.
- https://openclipart.org/detail/68413/database.
- Allan Foster, March 24, 2010, https://www.flickr.com/photos/foshydog/4461474358.
- Ricky Tang, November 12, 2010, https://www.flickr.com/photos/rickytang/5191447976.