
# Software Architecture

Dr. Hoger Mahmud | 2022

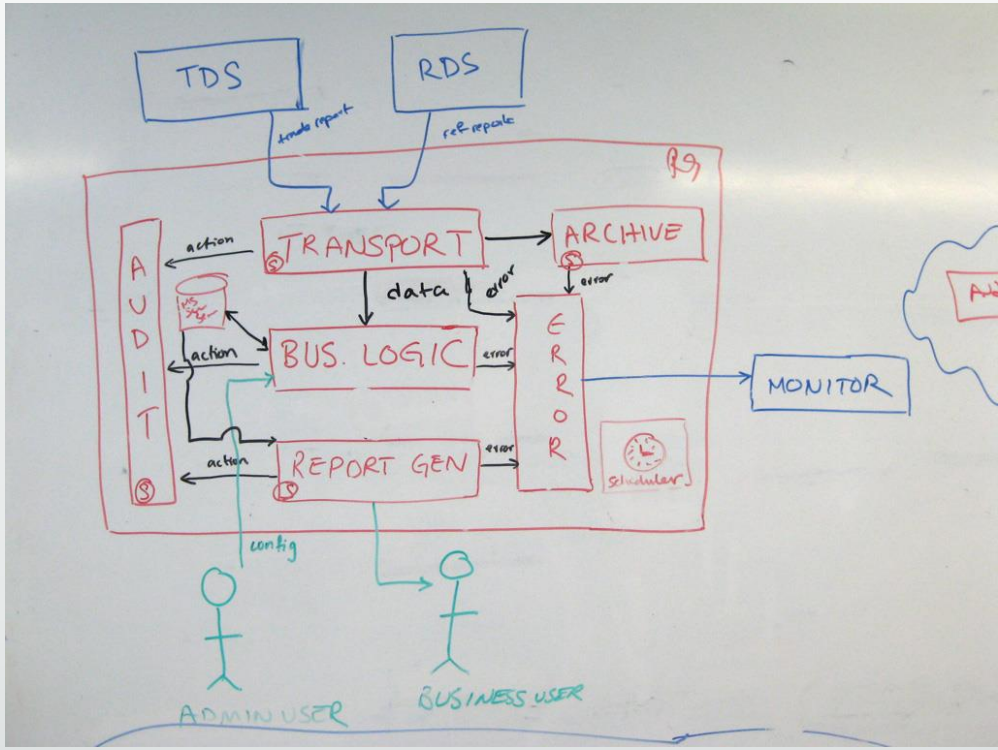
AUIS IT DEPARTMENT

# Content

- What is Software Architecture
- Architectural styles
- Architectural patterns
- Why styles and patterns
- Examples of styles and patterns

# What is software architecture

- **Software architecture** is, simply, the organization of a system. This organization includes:
  - All components
  - How they interact with each other
  - The environment in which they operate
  - The principles used to design the software



# Architectural styles

- **Architectural Styles** is the logical view of software architecture which deals with decomposing the software system into logical components that supports functional and non-functional (quality) requirements.

## Classification of Architectural Styles

| Type | Description |
|---|---|
| Data-Centered | Systems that serve as a centralized repository for data, while allowing clients to access and perform work on the data. |
| Data Flow | Systems oriented around the transport and transformation of a stream of data. |
| Distributed | Systems primarily involve interaction between several independent processing units connected via a network. |
| Interactive | Systems that serve users or user-centric systems. |
| Hierarchical | Systems where components can be structured as a hierarchy (vertically and horizontally) to reflect different levels of abstraction and responsibility. |

# Architectural patterns

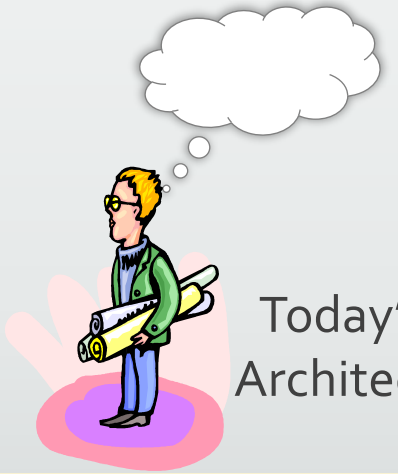
- An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.
- **Common architectural patterns (Implementation view)**
- Layered pattern
- Client-server pattern
- Master-slave pattern
- Pipe-filter pattern
- Broker pattern
- Peer-to-peer pattern
- Event-bus pattern
- Model-view-controller pattern
- Blackboard pattern
- Interpreter pattern

# Why architectural styles and patterns

- Architectural styles and architectural patterns provide generic, reusable solutions that can be easily understood.
- They can be easily applied to new problems requiring similar architectural features.

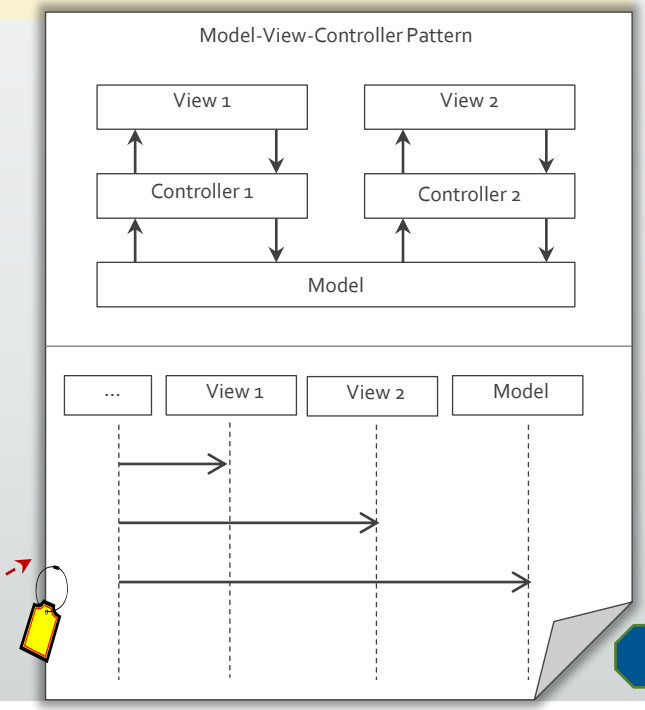
I need an interactive system, capable of displaying information from a data storage in multiple displays and different format!

Customer specifies what he needs!



Today's Architect!

Once a pattern is identified, Architects can always refer to a pattern catalog to find documented details about it!




Software architected this way exhibit certain quality properties

# Data-Centered Systems

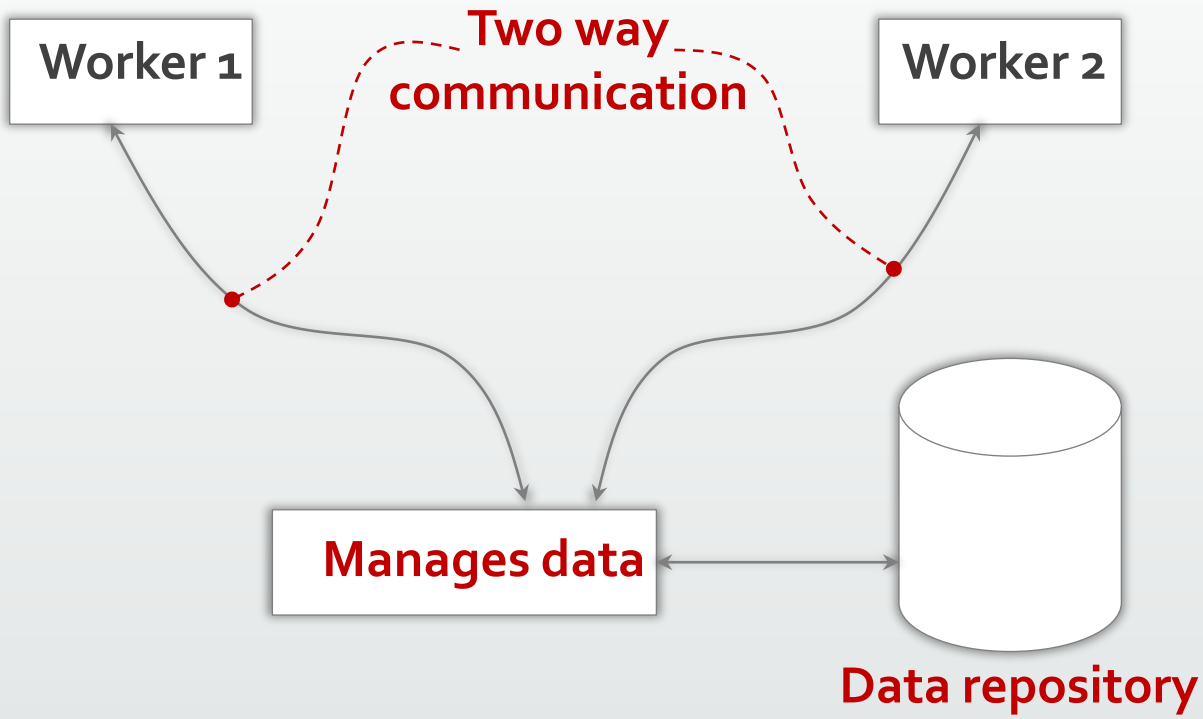
- **Data-centered systems** are systems primarily decomposed around a main central repository of data. These include:

## Data management component

- The data management component controls, provides, and manages access to the system's data.

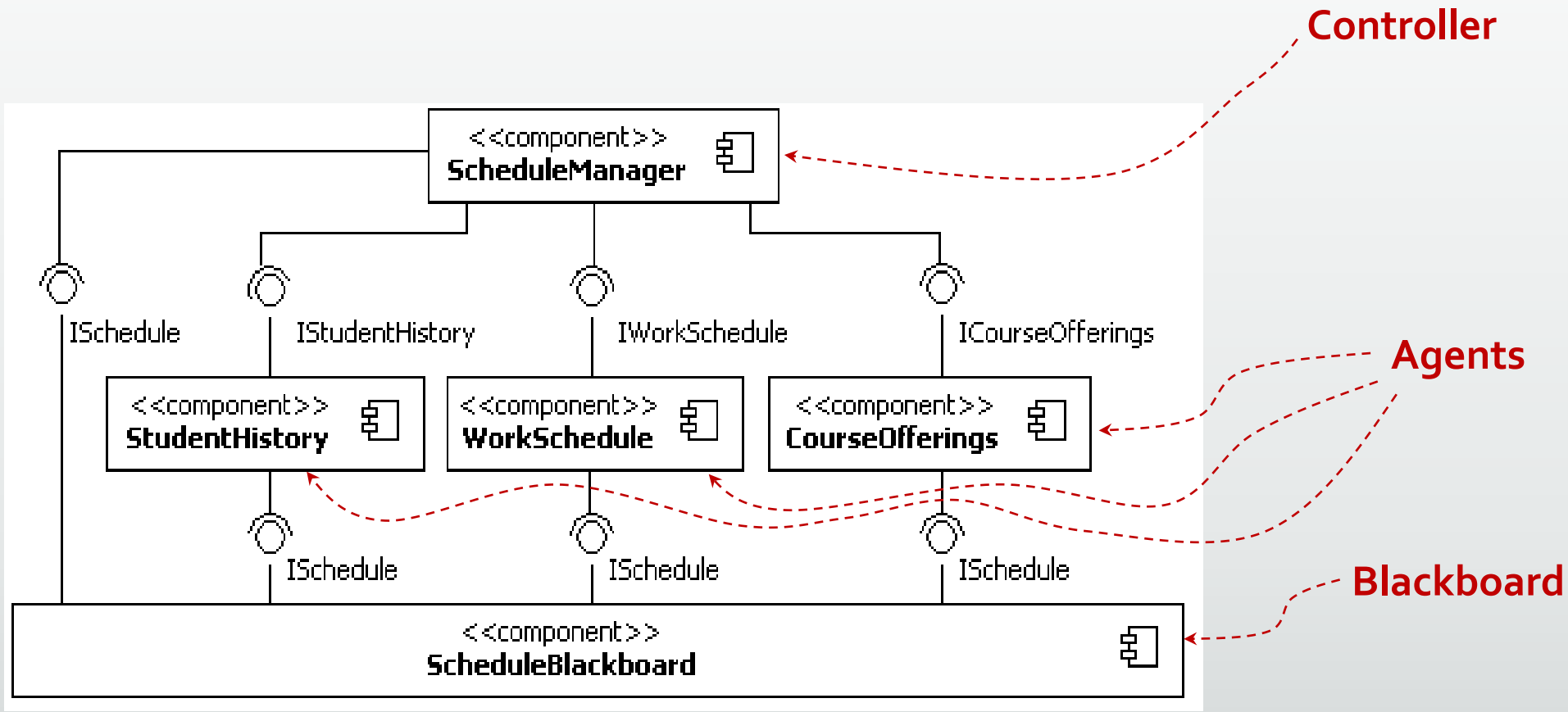
## Worker components

- Worker components execute operations and perform work based on the data.



# Blackboard Pattern implementation structure

- A common architectural pattern for data-centered systems is the Blackboard Pattern.



# Blackboard Architectural Pattern behavior

Client requests a schedule

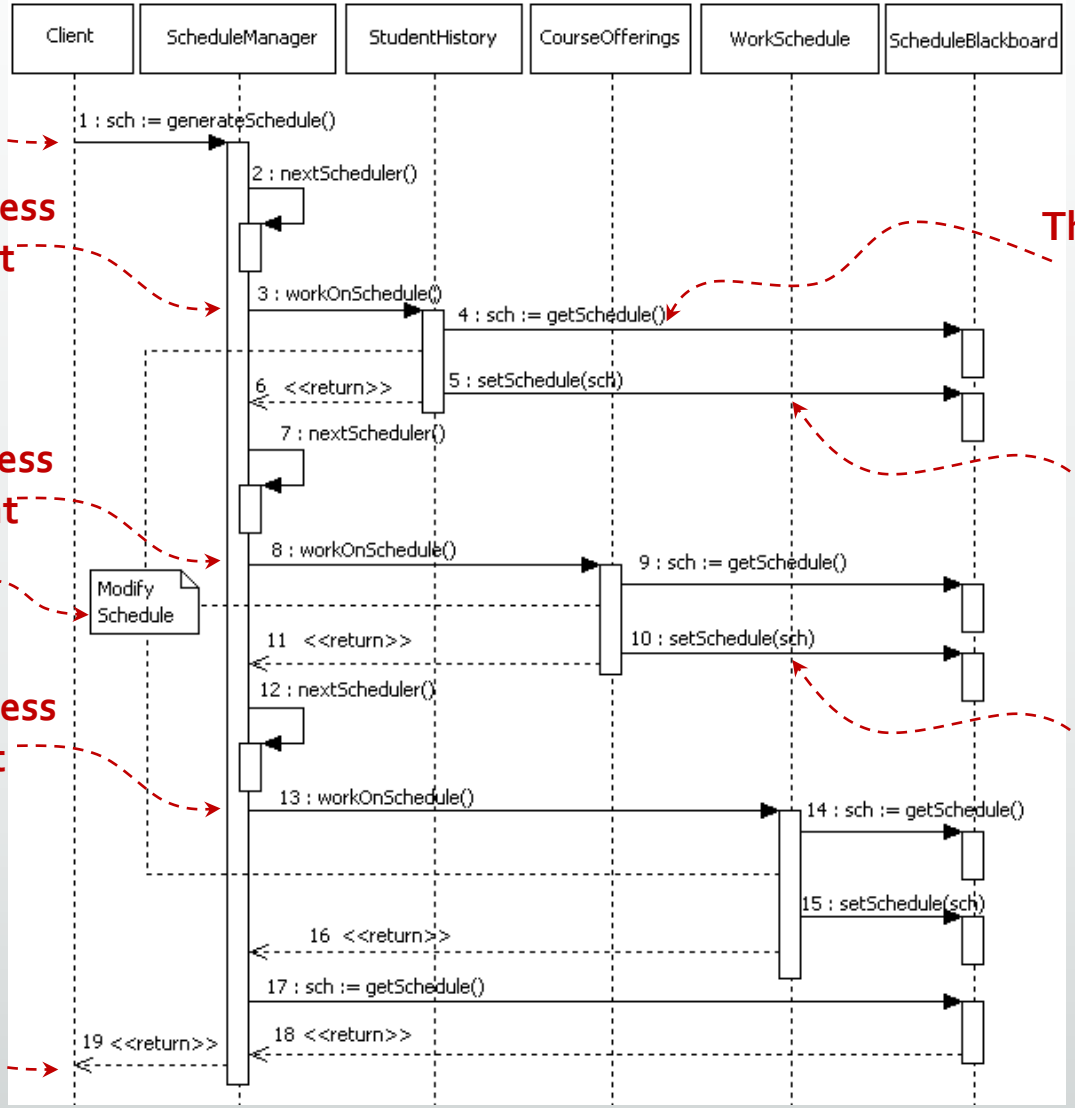
Controller grants schedule access to the Student History agent

Controller grants schedule access to the Course Offerings agent

This note provides important information!

Controller grants schedule access to the Work Schedule agent

Client receives an optimized schedule




The Student History agent retrieves the data

The Student History agent modifies the schedule and stores the results back

The Course Offerings agent retrieves, modifies, and stores the schedule back

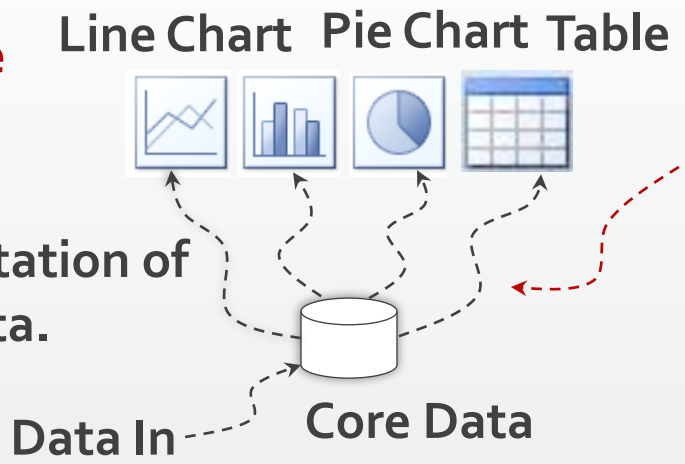
# Styel: Interactive Systems

- Interactive systems support user interactions, typically through user interfaces.
- The mainstream architectural pattern employed in most interactive systems is the Model-View-Controller (MVC).

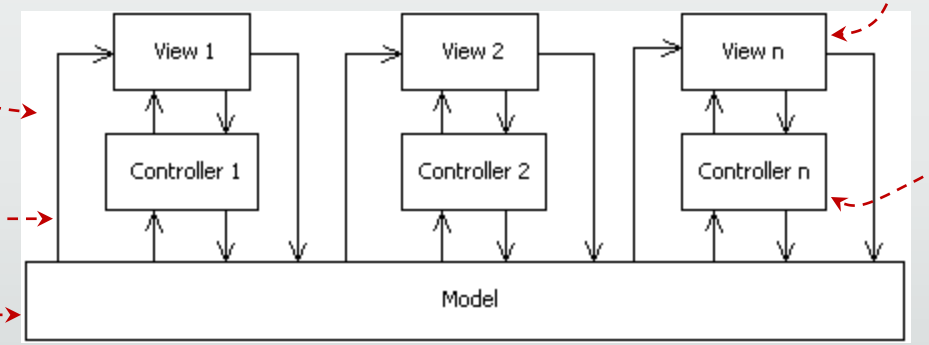
| Component | Description |
|---|---|
| Model | Component that represents the system's core, including its major processing capabilities and data. |
| View | Component that represents the output representation of the system (e.g., graphical output or console-based). |
| Controller | Component (associated with a view) that handles user inputs. |

# Model-View-Controller Architectural Pattern

Consider the popular example where data needs to be represented in different formats




Box-and-line diagram of the MVC architectural pattern



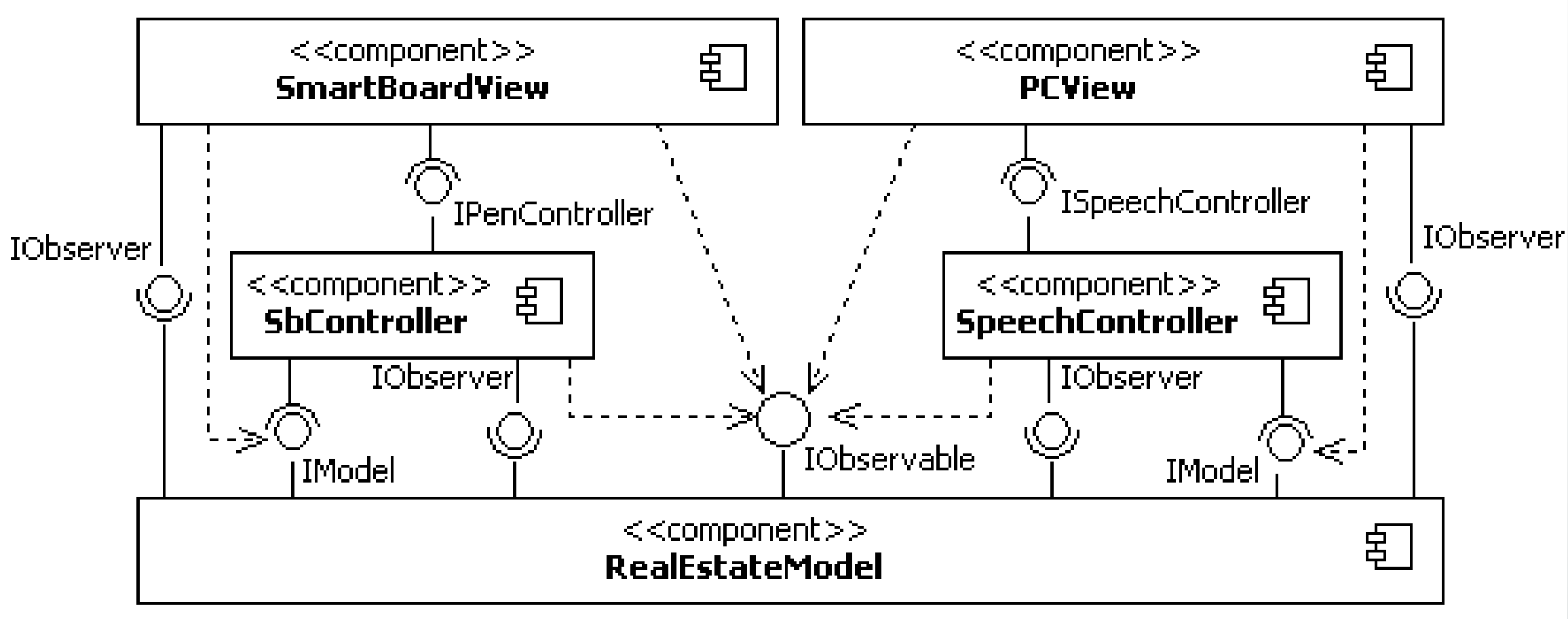

When Model changes, it updates its Views

<u>Model</u>: Encapsulates core data and functionality

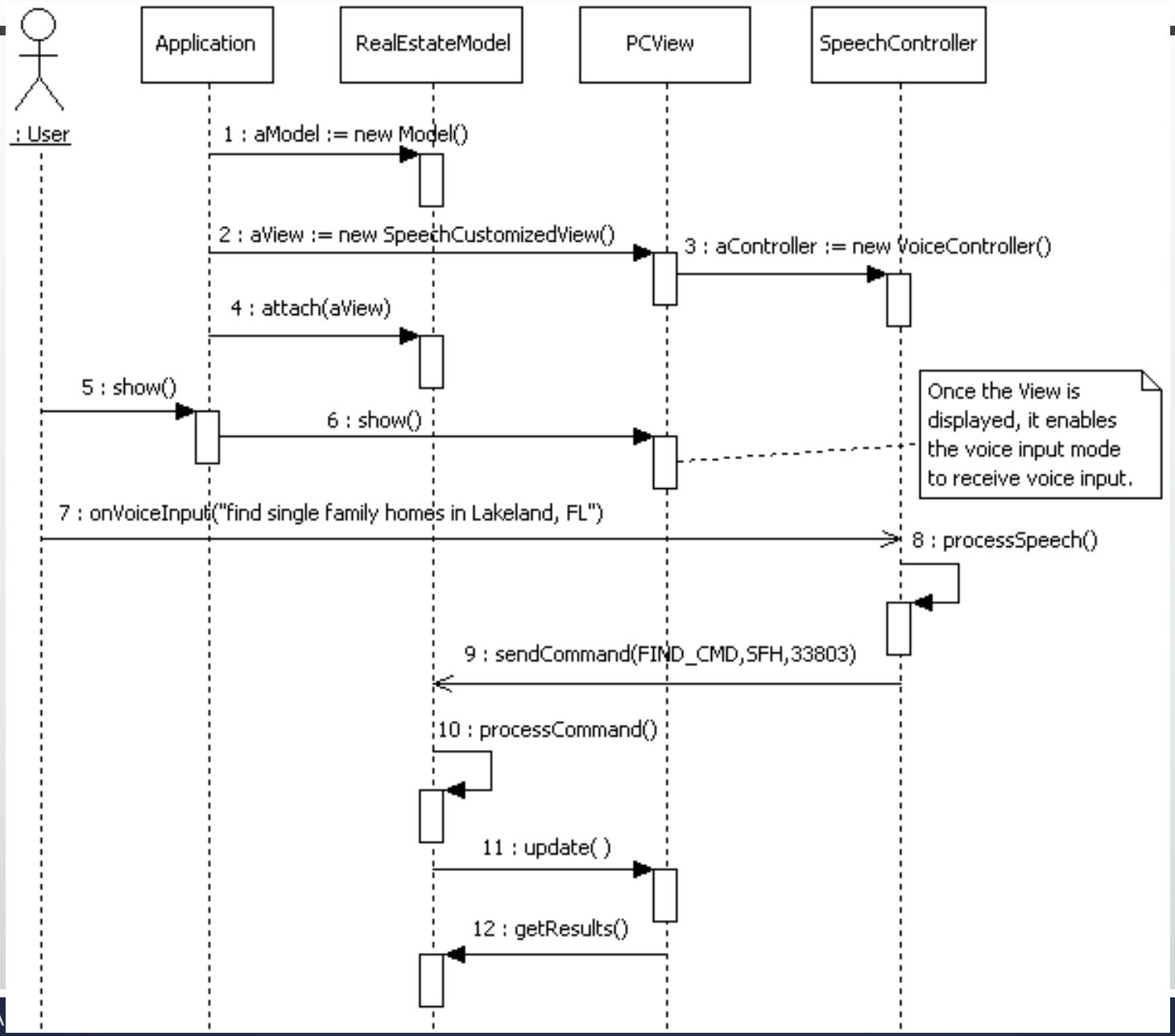
<u>View</u>: Display information to the user. Every View is associated with a Controller.

<u>Controller</u>: Receives input and translates it to service requests.

# Model-View-Controller Architectural Pattern



# Model-View-Controller Architectural Pattern




# References

- As specified in the syllabus