MVC

Overview

- The purpose of Model-View-Controller (MVC) MVC is to decompose the whole system into three sub-systems (modules) that are Model, View, and Controller.
- Each module in the MVC architecture has its own responsibility. Project team members with different expertise can work more efficiently in their own areas.
- For example, graphic professionals work on presentation of GUI interface module, programmer professionals work on input processing such as authentication, flow logic, job dispatching in the Controller module, and data processing and data base professionals can focus on the Model module to provide all the data the Web application needs.

- The Controller takes the inputs and does the authorization and authentication checking, dispatches incoming requests to the corresponding module or sub-module in Model by instantiating new instances of data sources in Model module and calling the methods provided by the data source objects.
- The data source forwards the controls to a specific representation module and lets that module be in charge of rendering the data retrieved from the data source objects. The data-Model module may generate events to notify the Model listeners when the data is changed.
- The View module listens to the data-Model module as its event listener. When such an event occurs the View module needs to update its presentation views.

Introduction

- Model-View-Controller ("MVC") is the recommended architectural design pattern for interactive applications
- MVC organizes an interactive application into three separate modules:
  - one for the application model with its data representation and business logic,
  - the second for views that provide data presentation and user input, and
  - the third for a controller to dispatch requests and control flow.

INTRODUCTION

- In programming, there is a term called “design patterns” that allow the developers to share their problems and solutions that benefit everyone.
- A design pattern is capable to identify the problem’s definition and context, a possible solution, and the solution’s consequences.
- Christopher Alexander, the initiator of the term “design pattern” in software engineering has stated that each pattern is divided into 3 part rule which expresses a relation between a certain context, a problem, and a solution.
- Hence, Design patterns provide reusable solution to recurring problems when developing software within a particular context.

INTRODUCTION

- Here, the Java Enterprise Edition uses the MVC (Model – View – Controller) as a design pattern used in most of the enterprise applications.
- The MVC architecture divides the web – based application into 3 parts: the Model, the View and the Controller.
- The pattern used separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes.
- Model : defines the lowest level of the pattern maintains data.
- View : displays all or a portion of the data to the user.
- Controller : controls the interactions between the model and view using software code.

CONTINUE

- Most Web-tier application frameworks use some variation of the MVC design pattern
- The MVC (architectual) design pattern provides a host of design benefits

MODEL

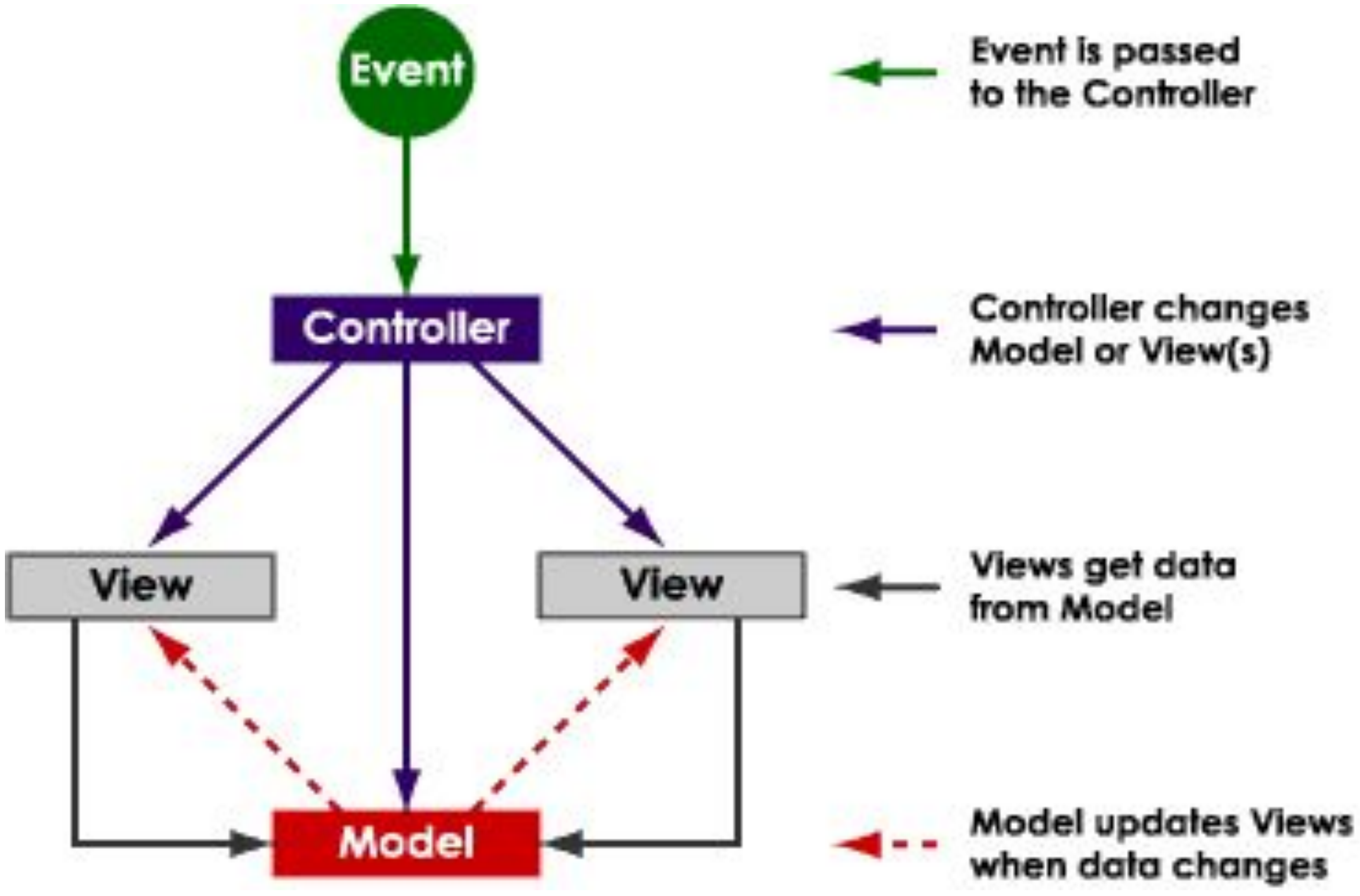
- The Model's responsibilities
  - Provide access to the state of the system
  - Provide access to the system's functionality
  - Can notify the view(s) that its state has changed

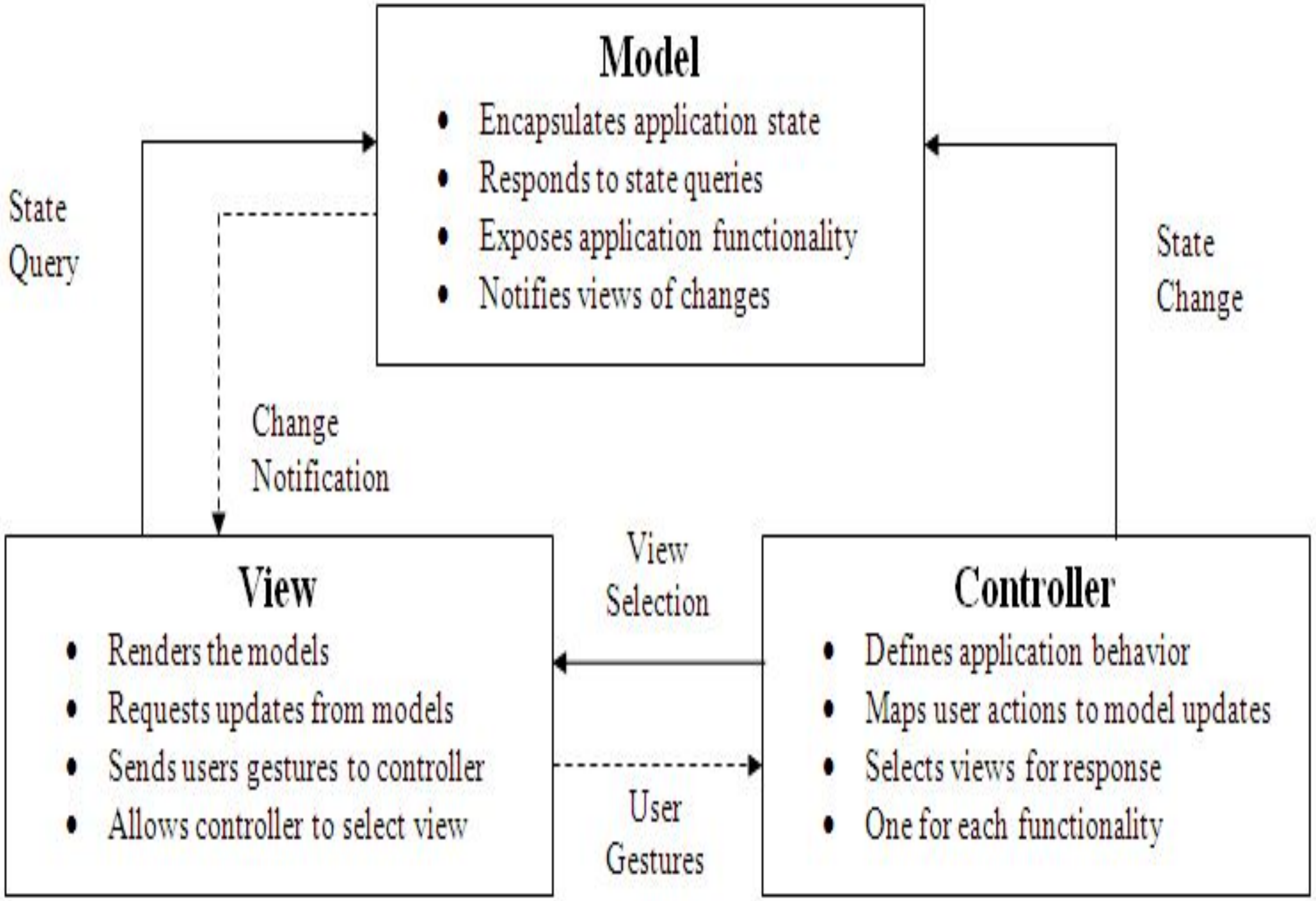
View

- The view's responsibilities
  - Display the state of the model to the user
- At some point, the model (a.k.a. the observable) must registers the views (a.k.a. observers) so the model can notify the observers that its state has changed.

Controller

- The controller's responsibilities
  - Accept user input
    - Button clicks, key presses, mouse movements, slider bar changes
  - Send messages to the model, which may in turn notify it observers
  - Send appropriate messages to the view
- In Java, listeners are controllers.


Event
Controller
View
View
Model
Event is passed to the Controller
Controller changes Model or View(s)
Views get data from Model
Model updates Views when data changes


Model
Encapsulates application state
Responds to state queries
Exposes application functionality
Notifies views of changes
State Query
State Change
Change Notification
View
Renders the models
Requests updates from models
Sends users gestures to controller
Allows controller to select view
View Selection
User Gestures
Controller
Defines application behavior
Maps user actions to model updates
Selects views for response
One for each functionality

Benefits

- Clarity of design
  - easier to implement and maintain
- Modularity
  - changes to one don't affect the others
  - can develop in parallel once you have the interfaces
- Multiple views
  - games, spreadsheets, powerpoint, Eclipse, UML reverse engineering, ….

Types of MVC (MVC Type – I)

- Here the view and the controller exist as one entity.
- In terms of implementation, in the page centric approach the controller logic is implemented within the view i.e. with JavaEE, it is JSP, for all the tasks of the controller like extracting HTTP request parameters, the business logic can be called (implemented in JavaBeans, if not directly in the JSP), and handling of the HTTP session is embedded within JSP using scriptlets and JSP action tags.

Types of MVC (MVC Type – II)

- The Type – I has the limitations of its lack of maintainability.
- With controller logic inside JSP using scriptlets, the code can get out of control very easily.
- Hence, to overcome the problems of maintainability and reusability, the controller logic can be moved into a servlet and the JSP can be used for its real implementation – the view component.
- Hence, using controller logic within a servlet, the MVC type – II design pattern can be implemented.

- The **Major difference** between type – I and type – II is where the controller logic is embedded in JSP in type – I and type – II it's moved to servlet.