Architectural Patterns

Marek Rychlý
rychly@fit.vutbr.cz

Brno University of Technology
Faculty of Information Technology
Department of Information Systems

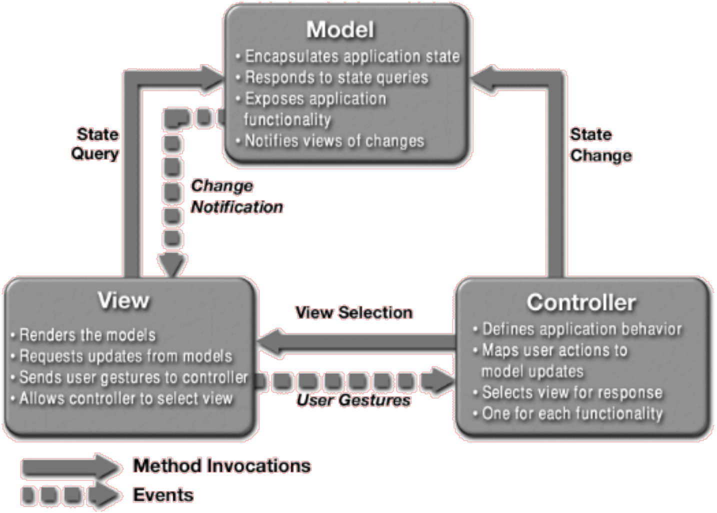
Information Systems Analysis and Design (AIS)
14 November 2019

Outline

1. Architectural Patterns
   - Model-View-Controller (MVC) and Model-View-Presenter (MVP)
   - Presentation-Control-Entity-Mediator-Foundation (PCMEF)
2. Patterns for Web Applications
   - Web MVC Architecture, Components, and Design Patterns
   - Web MVC in Java EE Applications
   - Model-View-Presenter (MVP), eXtensible Web Architecture (XWA)

Dependencies in Model-View-Controller (MVC)

(the events come from the observer design pattern, indirect dependencies)

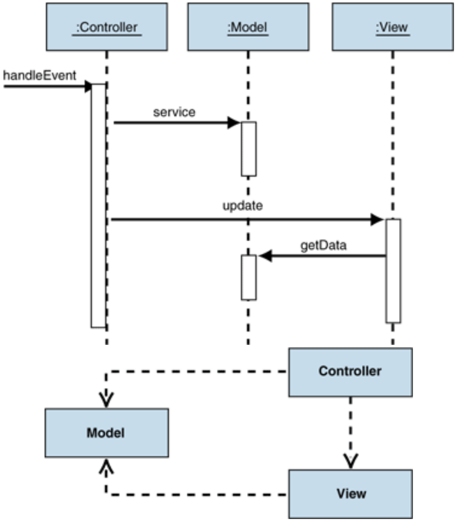

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view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

(adopted from “Java SE Application Design With MVC” by Robert Eckstein)

Model-View-Controller (MVC)

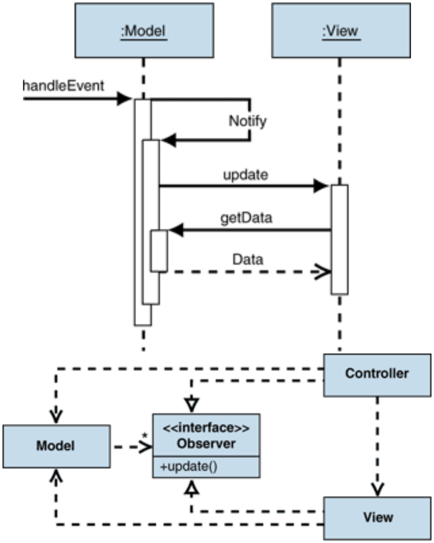
- Trygve Reenskaug, a Smalltalk developer at Xerox PARC in 1979.
  (to decouple data access & business logic from the manner in which it is displayed)
- Three architectural layers:
  - **model** represents data and the rules that govern access to and updates of this data;
    (a domain layer with domain object including data processing logic)
  - **view** renders the contents of a model, presents the data;
    (if the model data changes, the view must update its presentation as needed)
  - **controller** translates the user's interactions (e.g., GUI events, HTTP requests, etc.) with the view into actions that the model will perform.
    (a controller may also select a new view to present back to the user)
- To approaches to update data presented in the view which
  - registers itself with the model for change notifications (**push** model),
  - or is responsible for calling the model when it needs to retrieve the most current data (**pull** model).

MVC Pull Model (Passive View)

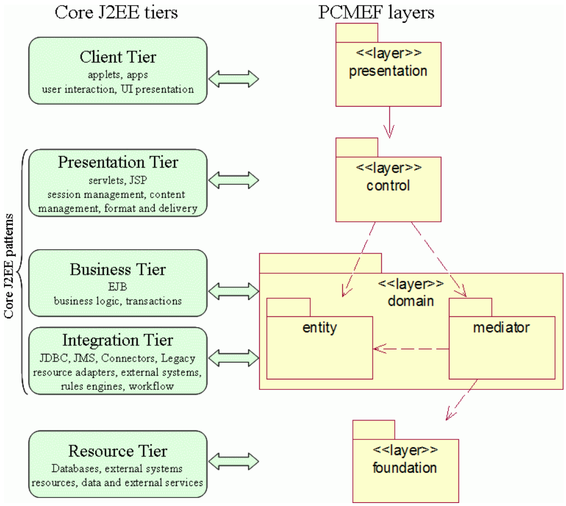


(adopted from “Model-View-Controller” by Microsoft)

MVC Push Model (Active View)



Presentation-Control-Entity-Mediator-Foundation
(PCMEF architectural pattern)


Core J2EE tiers
PCMEF layers
Client Tier
applets, apps
user interaction, UI presentation
Presentation Tier
servlets, JSP
session management, content management, format and delivery
Business Tier
EJB
business logic, transactions
Integration Tier
JDBC, JMS, Connectors, Legacy resource adapters, external systems, rules engines, workflow
Resource Tier
Databases, external systems
resources, data and external services
Core J2EE patterns
<<layer>> presentation
<<layer>> control
<<layer>> domain
entity
mediator
<<layer>> foundation

(adopted from "Practical Software Engineering" by Maciaszek&Liong)

Principles of PCMEF (1)

(according to “Round-trip architectural modelling” by Maciaszek)

DDP	Downward Dependency Principle (the main dependency structure is top-down where objects in higher layers depend on objects in lower layers, so the lower layers are more stable than the higher layers)
UNP	Upward Notification Principle (there is a low coupling in bottom-up communication between layers achieved by using asynchronous communication based on event processing where objects in higher layers act as subscribers (observers) to state changes of objects (publishers) in lower layers)
NCP	Neighbour Communication Principle (a package can only communicate directly with its neighbour package; message passing between non-neighbouring objects uses delegation)
EAP	Explicit Association Principle (associations are established on all directly collaborating classes documenting their communication in UML class models)

Principles of PCMEF (2)

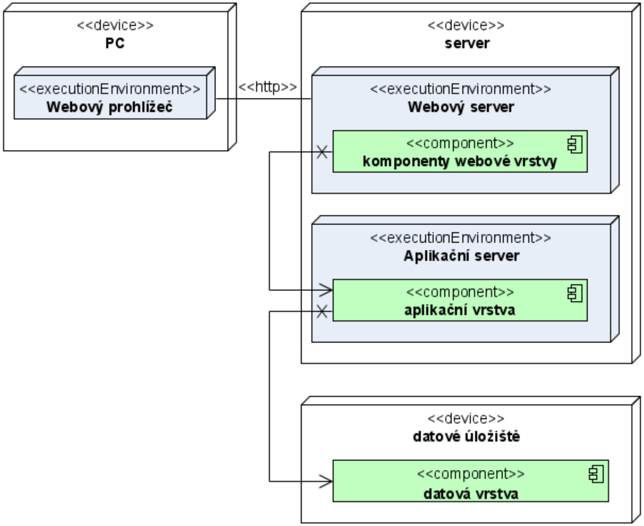
(according to “Round-trip architectural modelling” by Maciaszek)

CEP	Cycle Elimination Principle (circular dependencies between layers, subsystems and classes within subsystems are resolved by placing offending classes in a new subsystem/package created specifically for the purpose or by forcing one of the communication paths in the cycle to communicate via an interface)
CNP	Class Naming Principle (each class name is prefixed with the first letter of the subsystem name; each interface name is prefixed with the letter “I” followed by the first letter of the subsystem name)
APP	Acquaintance Package Principle (there is a special “acquaintance” subsystem consisting only of interfaces that an object passes in arguments to method calls, instead of concrete objects; while these interfaces can be implemented in any subsystem, their central declaration in the special subsystem effectively allows communication between non-neighbouring subsystems without their direct dependency)

Web Applications

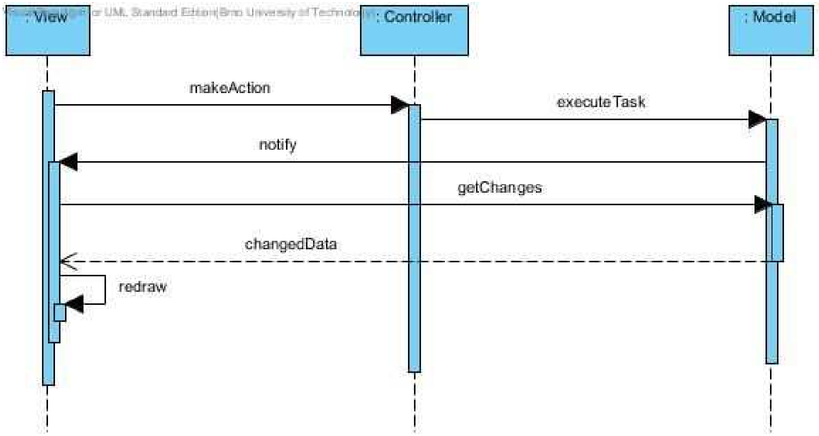
- There are Web applications with
  - Web UI – presentation-based application,
  - Web API – service-oriented applications.
- There are usually three or four tiers of client-server architecture:
  - a Web browser (a client, with or without a logic in HTML5),
  - a Web server (a presentation server),
  - an application server (can be integrated into the Web server),
  - a database server.
- Client-server interaction as request-reply via the HTTP protocol.
  (HTTP method such as GET/POST/. . . ; the state-less protocol; a text content)
  - + easy to deploy and to operate
    (it is not necessary to provide/update the client-side of an application)
  - – a Web server usually cannot push content/events to its clients
    (however, HTML5 introduced full-duplex WebSocket API for the server push)
  - – the statelessness does not allow to keep the context
    (however, HTTP/2 has stateful components for encryption, compression, etc.)

Deployment of Web Applications



(adopted from “Design Patterns for Web Applications” by Jan Dudek)

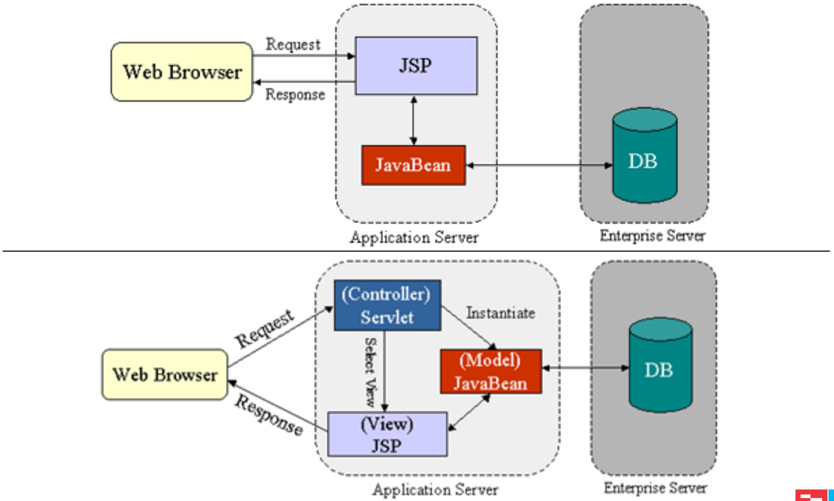
Interaction of Layers in MVC



(adopted from “MVC u webových aplikací” by Jaroslav Zendulka)

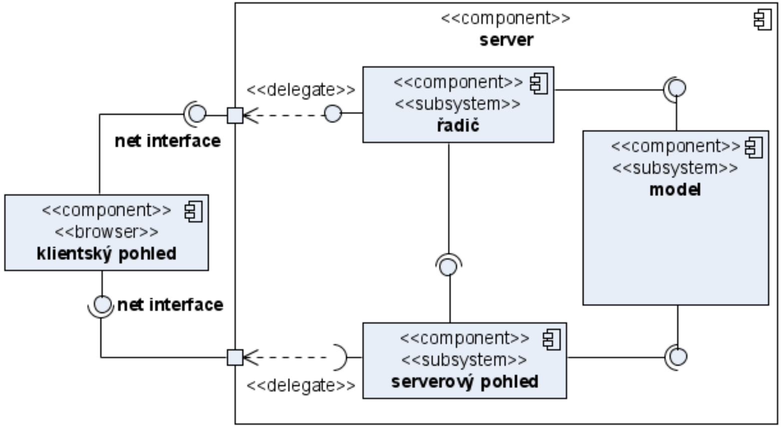
Web UI issue: a server-side model cannot notify a client-side view.
(MVC push model is not possible – there is HTTP between view and controller/model)

J2EE: JSP Model 1 & Model 2 Architecture



(adopted from "Servlets and JSP Pages Best Practices" by Qusay H. Mahmoud)

Web MVC: Client-side and Server-side Views



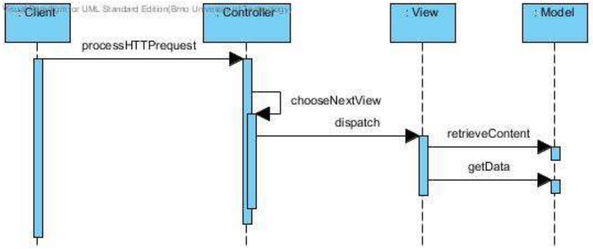
(adopted from “Design Patterns for Web Applications” by Jan Dudek)

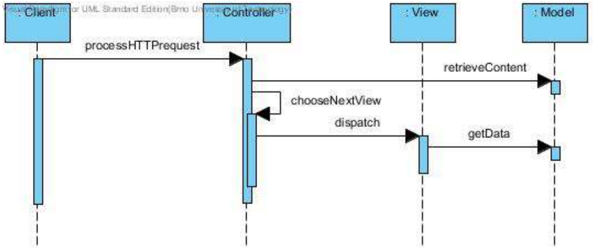
The server-side view is prepared and sent/mirrored to a client.
(there are two approaches how to prepare the server-side view from data in the model)

Web MVC Design Patterns

- It would be wrong to integrate UI and application logic into JSP.
  (against the SoC design principle – different concerns and development)
- Web MVC: a controller handles UI actions, a view presents data.
  (based on the actions, the controller can switch the view or modify its model)
  - Front Controller
    (as the initial point of contact for handling all related requests/control logic)
  - Application Controller/Dispatcher
    (as the custom initial access point for a request passing the front controller)
  - View Helper & Business Helper
    (to encapsulate view-processing logic & utilised business objects in a view)
- Two design patterns on how to prepare a response on the request:
  - Dispatcher View
    (to handle a request and generate a response, while managing limited amounts of business processing)
  - Service to Worker
    (to perform core request handling and invoke business logic before control is passed to the view)

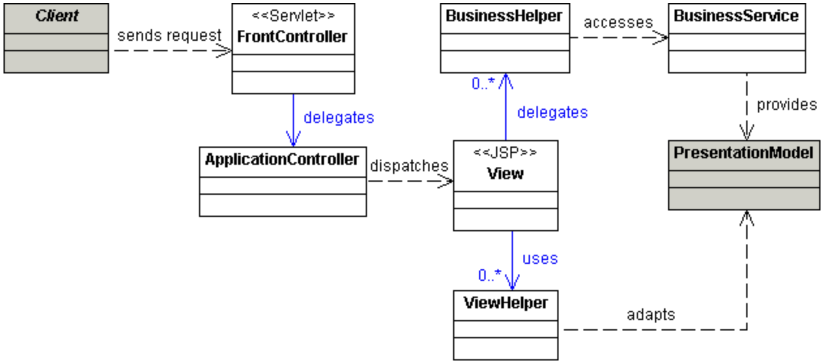
Dispatcher View and Service to Worker (Simplified)





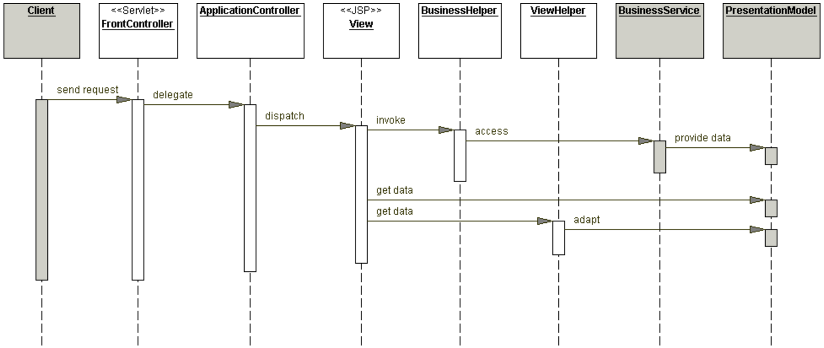
(adopted from “MVC u webových aplikací” by Jaroslav Zendulka)

Dispatcher View Classes



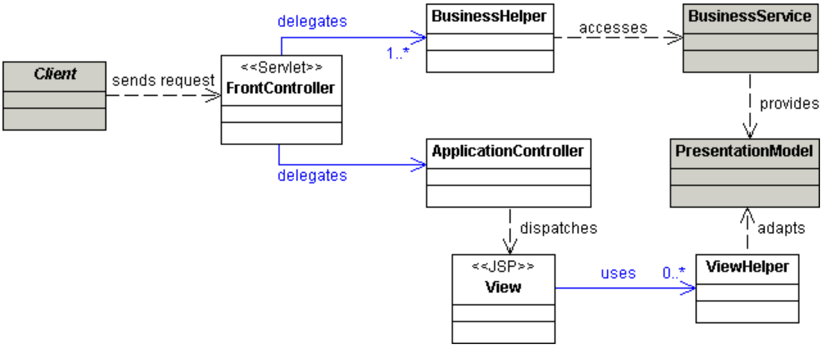
(adopted from “Core J2EE Pattern Catalog” by Oracle)

Dispatcher View Interaction



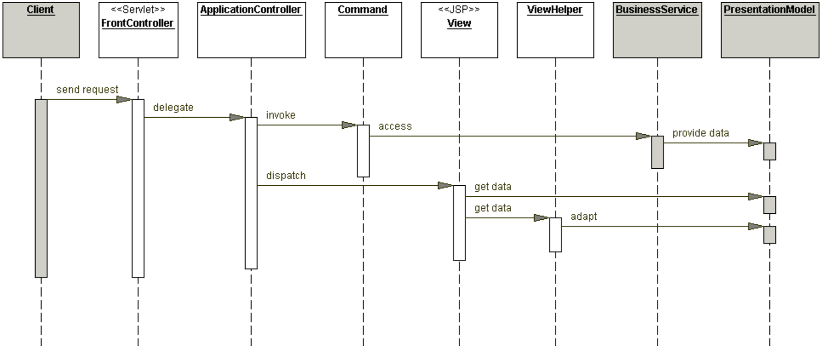
(adopted from “Core J2EE Pattern Catalog” by Oracle)

Service to Worker View Classes



(adopted from “Core J2EE Pattern Catalog” by Oracle)

Service to Worker Interaction

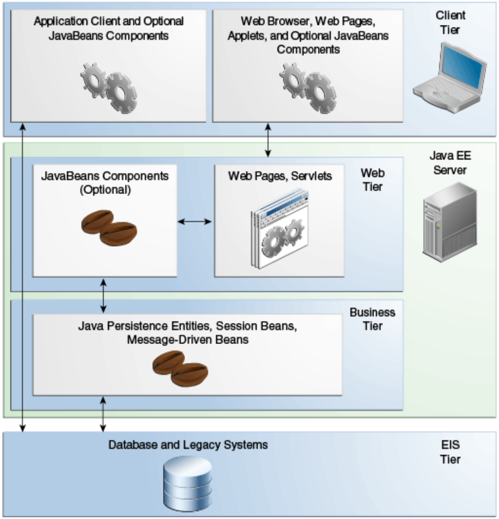


(adopted from “Core J2EE Pattern Catalog” by Oracle)

Web MVC Components

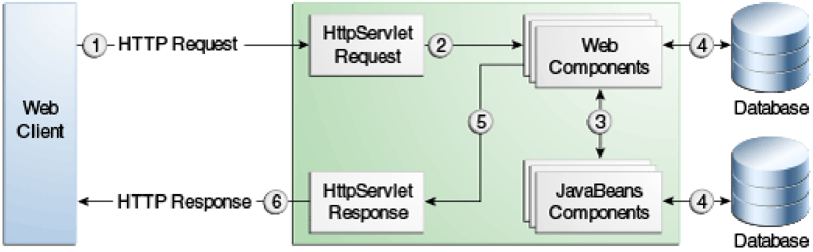
- Front Controller
  (as the initial point of contact for handling all related requests/control logic)
  - centralizes control logic that might otherwise be duplicated,
    (authentication, access control, validation, etc; can be delegated to helpers)
  - and manages the key request handling activities.
    (to invoke particular views, commands, etc; it depends on the design pattern)
- Application Controller/Dispatcher
  (as the custom initial access point for a request passing the front controller)
  - to manage views & their flow; to select the next view for the current;
  - in general MVC, this is integrated together with the from controller.
- View
  (as a server-side representation of the client-side view)
  - data for the presentation loaded from a model, w/ or w/o helpers
- View Helper & Business Helper
  (to encapsulate view-processing logic & utilised business objects in a view)
- Business Service
  (as a model abstraction representing a particular service request by a user)

Tiers of Java Web Applications


Application Client and Optional JavaBeans Components
Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
JavaBeans Components (Optional)
Web Pages, Servlets
Web Tier
Java EE Server
Java Persistence Entities, Session Beans, Message-Driven Beans
Business Tier
Database and Legacy Systems
EIS Tier

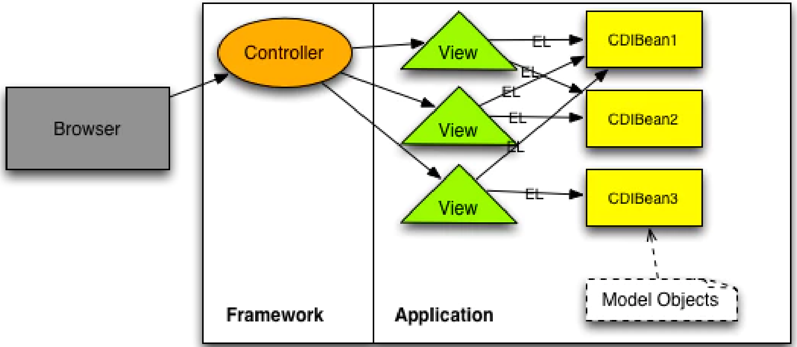
(adopted from “Java EE 8 – The Java EE Tutorial” by Oracle)

Web Tier in Java EE Applications



(adopted from “Java EE 8 – The Java EE Tutorial” by Oracle)

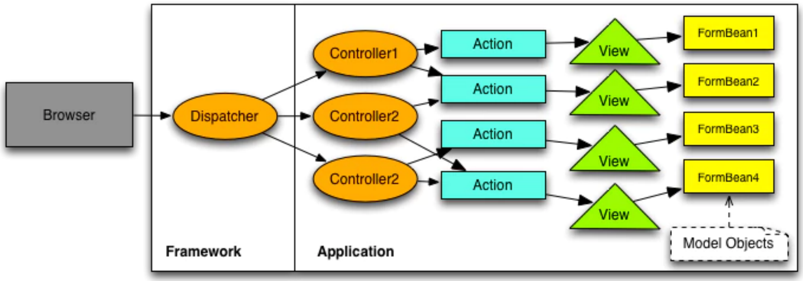
UI Component Oriented MVC (JSF in Java EE 5)



(adopted from “Why Another MVC?” by Ed Burns)

- Web UI rendered from a set of the framework components in HTML/JS.
- Automatic (state-full) request parameter processing by the framework. (including conversions and validations with respect to the developer configurations)
- Page centric – a developer provide views/pages and define their flow.

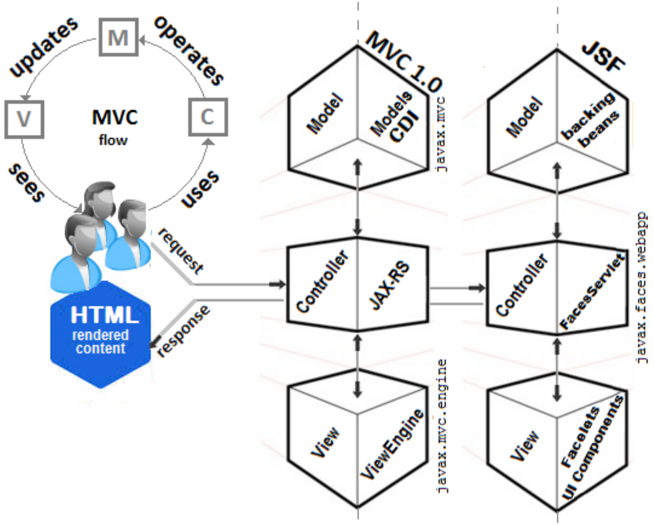
Action Oriented MVC (MVC 1.0 in Java EE 8)



(adopted from “Why Another MVC?” by Ed Burns)

- Web UI design is in the developers’ hands.
  (in a wide range of technologies, such as HTML5, JS, UI libraries, and so on)
- Manual (state-less) request parameter processing.
- Request centric – a developer implements also the controller.
  (the controller dispatches to a specific action, based on information in the request)

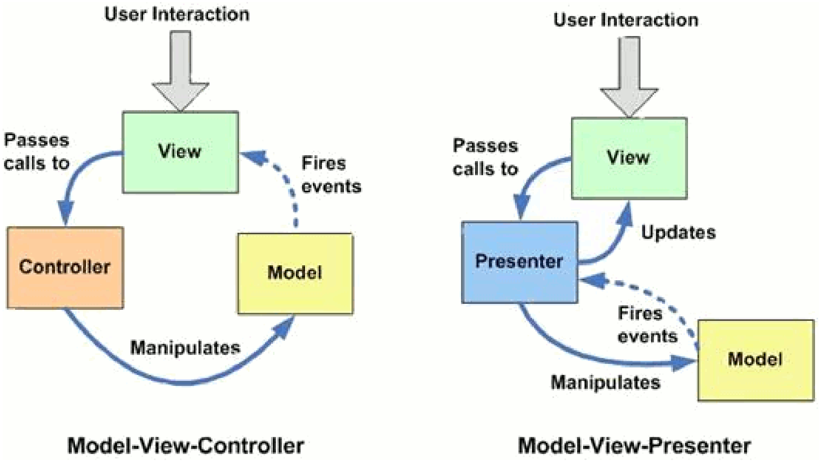
Java MVC 1.0 and JSF Layers



(adopted from "Introduction to the New MVC 1.0 (Ozark RI)" by Leonard Anghel)

Model-View-Presenter (MVP)

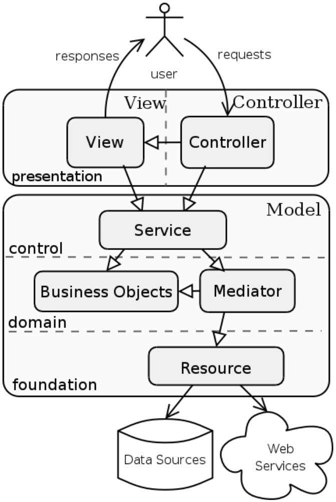
for Web-based applications



(adopted from “ASP.NET Patterns every developer should know” by Alex Homer)

eXtensible Web Architecture (XWA)

(an MVC/PCMEF architectural pattern for Web-based applications)



(adopted from “Architectural design of modern web applications” by Madeyski&Stochmialek)

Thank you for your attention!

Marek Rychlý
<rychly@fit.vutbr.cz>