Modele de proiectare a aplicațiilor de întreprindere

Cursul 8 - 9

Sumar

- Modele arhitecturale
- Modele de proiectare pentru aplicații Web

Modele arhitecturale

Modele arhitecturale

- Organizarea sistemelor informatice
- Abordare la nivel înalt, global
  - Componente
  - Mecanisme specifice sistemului
  - Comunicare
- Aspecte urmărite
  - Scalabilitate
  - Partiționarea sistemului
  - Protocoale
  - Interfețe

Modele arhitecturale (exemple)

- Arhitectură stratificată
- Arhitectură condusă de evenimente
- Arhitectură de tip micro-kernel
- Arhitectură bazată de microservicii
- Arhitectură bazată pe cloud

Arhitectura stratificată

Prezentare (interfața utilizator)
Logica organizațională
Servicii
Persistență
Baze de date

Modele de proiectare pentru aplicații Web

- Model View Controller (MVC)
- Model View Presenter (MVP)
- Model View ViewModel (MVVM)
- Model View Template (MVT)
- Front Controller
- Page Controller
- Template View
- Intercepting Filter
- Transform View
- Application Controller
- Two Step View

Model View Controller (MVC)

Necesitate

- Separarea rolurilor
- Testabilitate
- Modularitate

Model View Controller

- Model arhitectural
- Nivelul de prezentare
- Interacțiunea cu interfața este distribuită în trei roluri distincte
- Este aplicată separarea rolurilor

Componente

- **Model**
  - Încapsulează datele și accesul la acestea
  - Gestionează starea aplicației
  - Notifică *View-ul* cu privire la modificările apărute
- **View**
  - Interfața utilizator
  - Logica prezentării
  - Gestionează interacțiunea cu utilizatorul
- **Controller**
  - Gestionează interacțiunile
  - Corelează acțiunile utilizatorului cu datele asociate
  - Conectează modelul și interfața

Tipuri

- Model pasiv
  - Modelul nu include suport pentru notificarea schimbărilor
- Model activ
  - Modelul permite notificarea componentelor abonate

Diagrama (model pasiv)

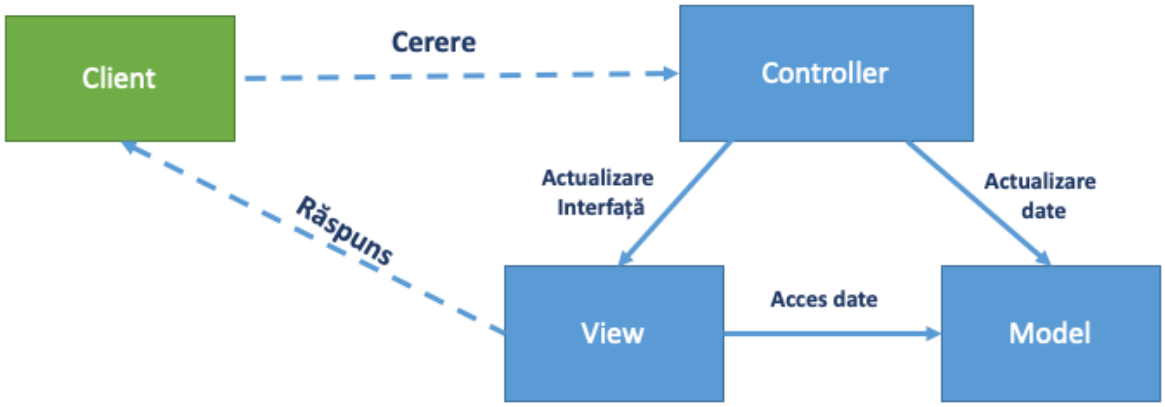
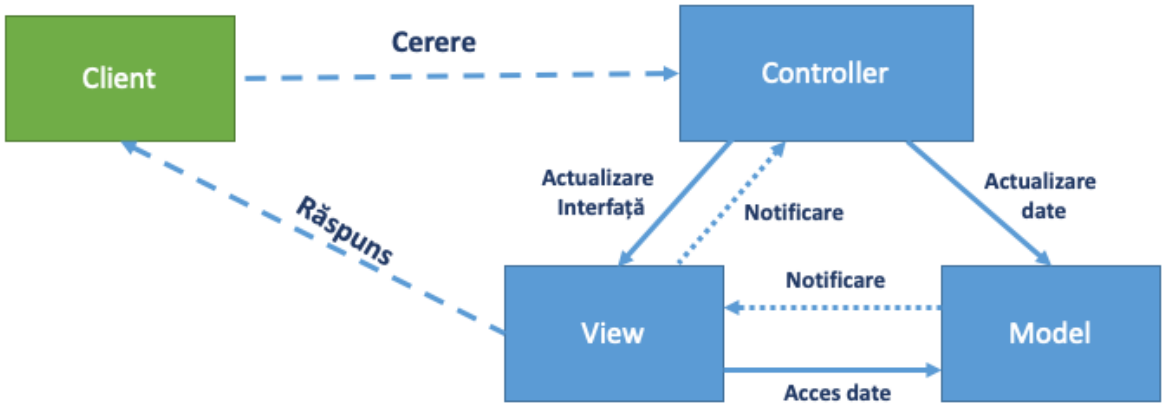


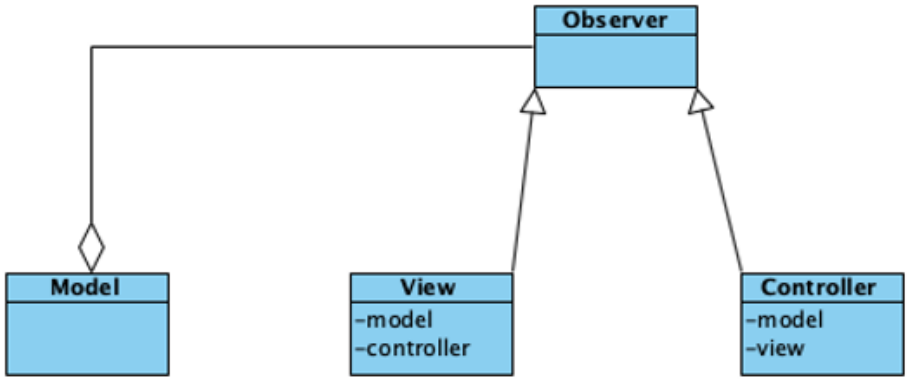
Diagrama (model activ)



Implementare

- View-ul recepționează un eveniment (ex. clic pe un buton)
- Este invocată metoda corespunzătoare din controler
- Controlerul accesează clase/metode din model
- Dacă modelul a fost modificat, se trimit notificări
- În urma notificărilor, view-ul este actualizat

Diagrama de clase (model activ)



Utilizare

- Se instanțiază modelul
- Se instanțiază view-ul, pe baza modelului
- Se adaugă obiectele de tip view la lista de notificări a modelului
- Se instanțiază controlerul, cu referințe la model și view

Implementări cunoscute

- ASP.NET MVC
- Rails
- Spring MVC

MVC

Avantaje

- Separarea interfeței utilizator de model
- Modelul poate fi testat independent
- Existența mai multor componente de interfață pentru același model

Dezavantaje

- Controlerul și view-ul sînt legate destul de puternic
- Complexitate ridicată pentru aplicațiile simple

Model-View-Presenter (MVP)

Model-View-Presenter (MVP)

- Model arhitectural
- Suport pentru automatizarea testării
- Componenta de tip View este punctul de intrare al aplicației

Tipuri

- View pasiv
  - View-ul nu se actualizează singur
  - View-ul nu este informat cu privire la modificările modelului
  - Presenter-ul gestionează legătura cu modelul
- Controler supervizor
  - View-ul poate interacționa direct cu modelul
  - Presenter-ul actualizează modelul
    - Modelul notifică view-ul

Diagrama (View Pasiv)

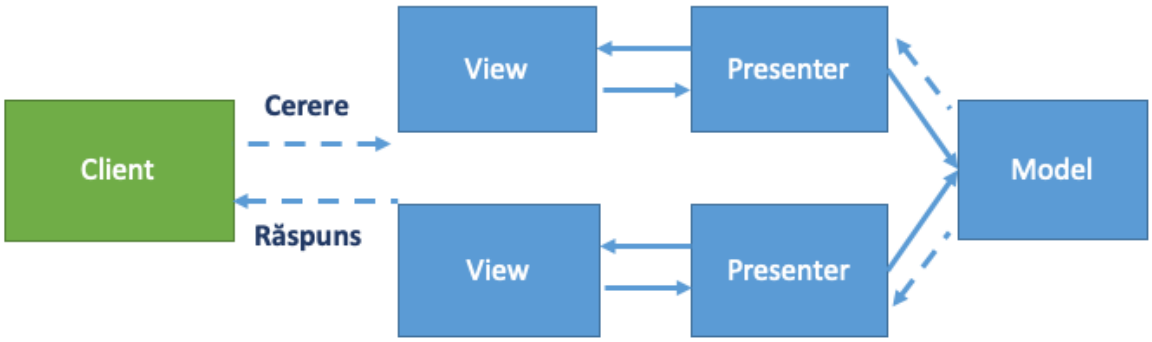
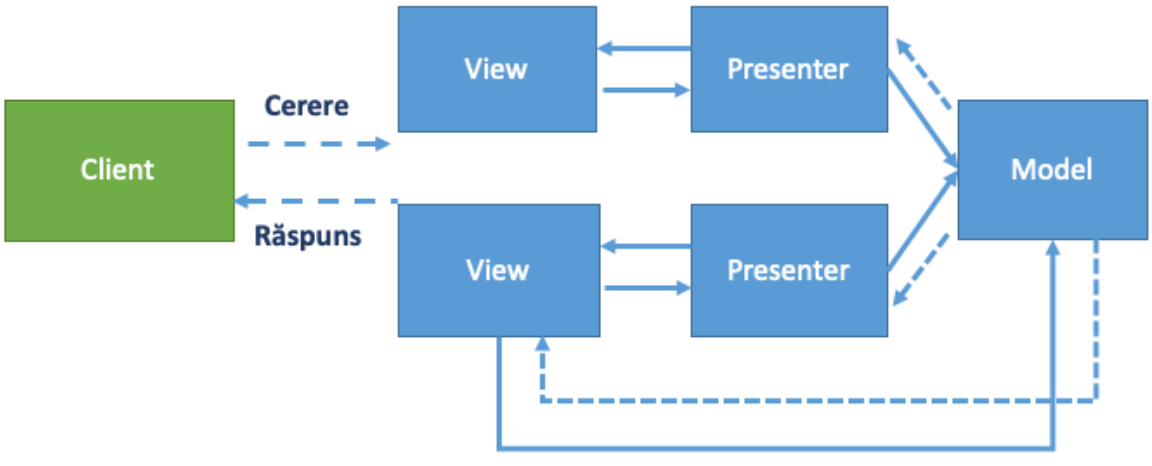


Diagrama (Controler supervizor)



Componente

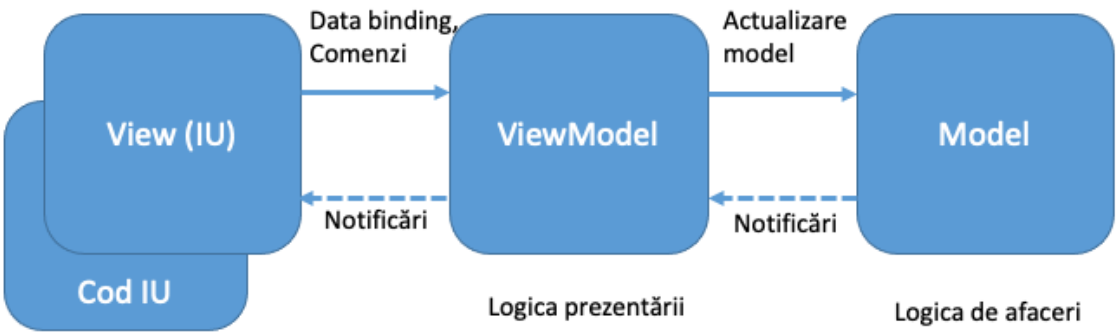
- **Model**
  - Nivelul datelor aplicației (acces la baze de date, preluare rețea etc.)
- **View**
  - Nivelul interfața utilizator
  - Afișează datele
  - Notifică **Presenter**-ul cu privire la acțiunile utilizatorilor
- **Presenter**
  - Legătura între date și interfața utilizator
  - Preia datele din model
  - Aplică logica interfeței utilizator
  - Gestionează starea View-ului
  - Reacționează la interacțiunea utilizatorilor cu View-ul

MVP

- Față de MVC, testabilitate mai mare la nivelul componentelor
- Cuplarea mai scăzută a componentelor

Model-View-ViewModel (MVVM)

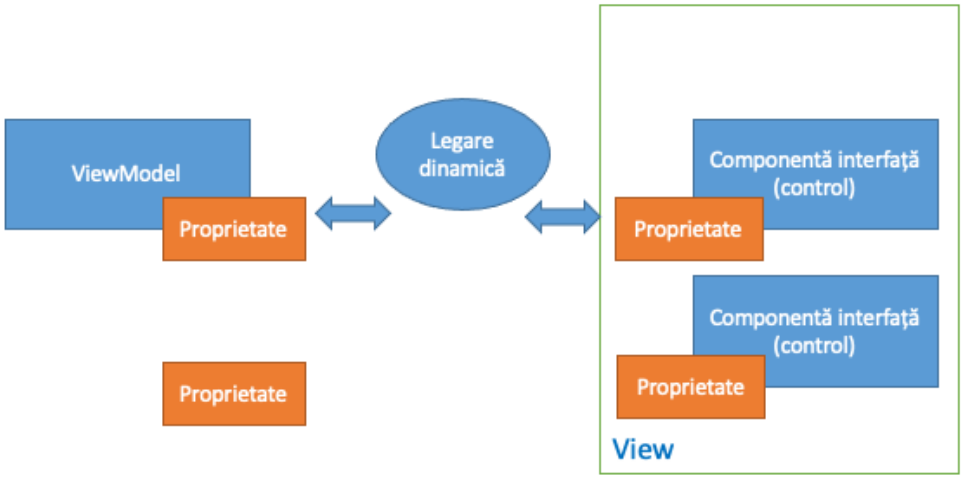
Diagrama



Componente

- **Model**
  - Datele aplicației (acces la baze de date, preluare rețea etc.)
- **View**
  - Interfața utilizator
- **ViewModel**
  - Legătura între date și interfață
  - Uzual, se utilizează legarea dinamică a datelor (*data binding*)
    - Proprietăți ViewModel
    - Proprietăți componente interfață utilizator
  - Generează evenimente la modificări ale datelor
    - Sursa de date sau din interfață

Legarea dinamică a datelor



Implementări cunoscute

- Silverlight
- Windows Presentation Foundation (WPF)
- Xamarin.Forms
- Android Data Binding

Implementări cunoscute (Web)

- Knockout

Model View Template (MVT)

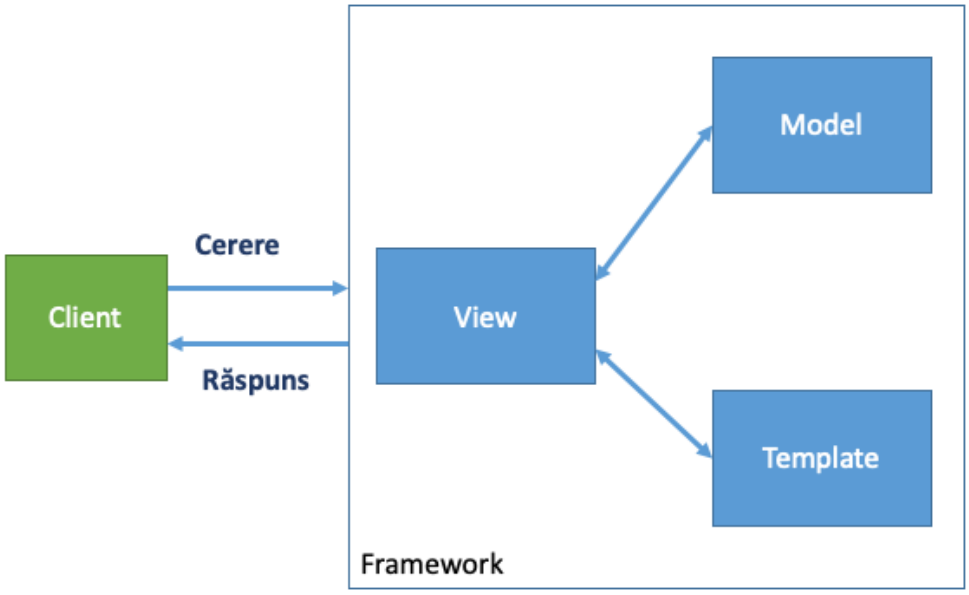
Model View Template

- Model arhitectural
- Interacțiunea cu interfața este distribuită în trei roluri distincte
- Framework-ul gestionează interacțiunea dintre componente
- Este aplicată separarea rolurilor

Componente

- **Model**
  - Încapsulează datele și accesul la acestea
  - Notifică *View-ul* cu privire la modificările apărute
- **View**
  - Logica aplicației
  - Interacționează cu modelul și
- **Template**
  - Nivelul de prezentare
  - Gestionează interacțiunea cu interfața aplicației

Diagrama



Implementări cunoscute

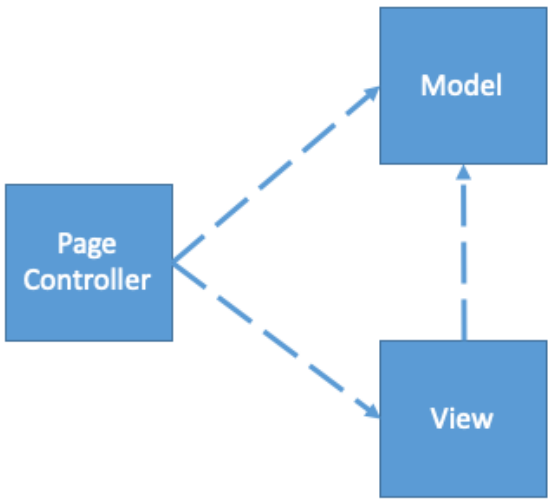
- Django

Page Controller

Page Controller

- Obiect desemnat să trateze o cerere pentru o anumită pagină sau acțiune într-un site Web
- Pentru fiecare pagină există un controler
- Controlerul poate fi implementat
  - script pe partea de server
  - pagină dinamică
- Utilizare pentru aplicații Web de complexitate medie

Diagrama



Componente

- **Page Controller**
  - Gestionează metodele HTTP GET și POST
  - Decide modelul și pagina care vor fi utilizate
- **Model**
  - Implementează logica domeniului
- **View**
  - Afișează conținutul HTML

Front Controller

Front Controller

- Acționează ca punct unic de intrare al unei aplicații Web
- Redirecționează cererile către diferite controlere de pagini
- Acționează în două etape
  - Gestiunea cererilor
  - Gestiunea comenzilor
- Variantă modificată a modelului de proiectare Mediator

Diagrama

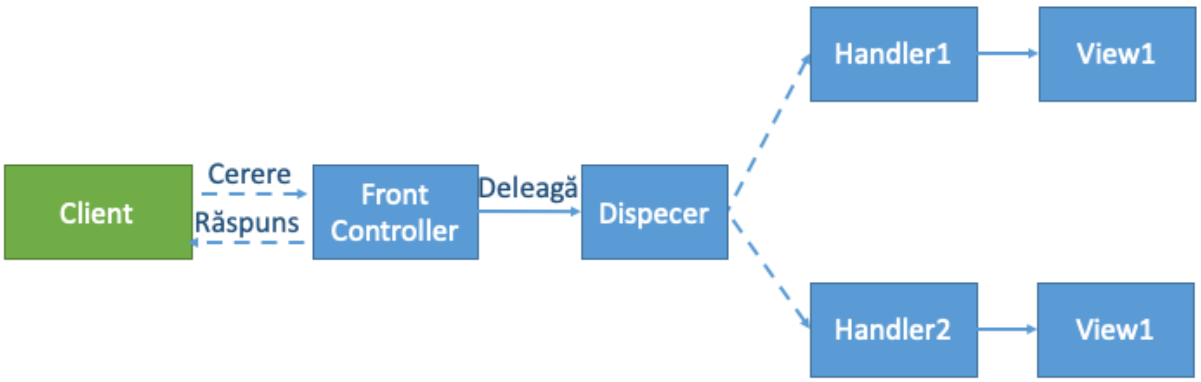
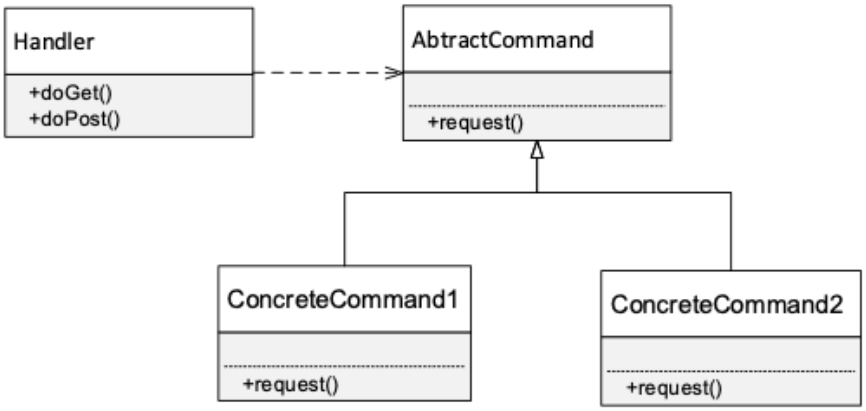


Diagrama de clase



Componente

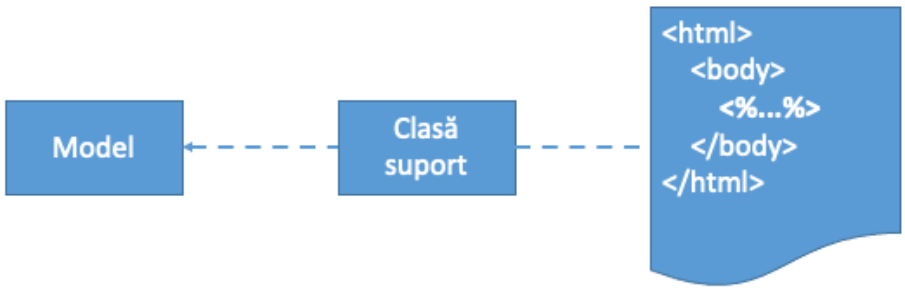
- **Handler**
  - Recepționează comenzile GET și POST
  - Analizează URL-ul și decide, static sau dinamic, care va fi comanda invocată
- **AbstractCommand**
  - Declară interfața pentru comenzile invocat de către Handler
- **ConcreteCommand1**, **ConcreteCommand2**
  - Comenzi concrete, implementate pe baza interfeței AbstractCommand

Template View

Template View

- Generează conținut HTML prin intermediul marcajelor specifice
- Suport pentru execuția condiționată și repetitivă
- Utilizare
  - ASP/ASP.NET
  - JSP
  - PHP
- Uzual, se utilizează împreună cu Page Controller în aceeași pagină
- Marcajele sînt interpretate de o componentă dedicată
  - este generat conținutul HTML

Diagrama



Componente

- Pagini care includ marcaje
  - Marcajele sînt asociate datelor dinamice
  - Datele dinamice sînt recepționate la invocarea paginii
- Obiecte suport
  - Preiau și furnizează valorile specifice paginilor
  - Încapsulează codul în cadrul claselor

Intercepting Filter

Probleme

- Clientul a fost autentificat?
- Clientul are o sesiune validă?
- Adresa IP a clientului este dintr-o rețea de încredere?
- Calea cererii încalcă vreo constrângere?
- Ce codificare folosește clientul pentru a trimite datele?
- Este suportat navigatorul Web?
- Trebuie decomprimat/decodificat conținutul?

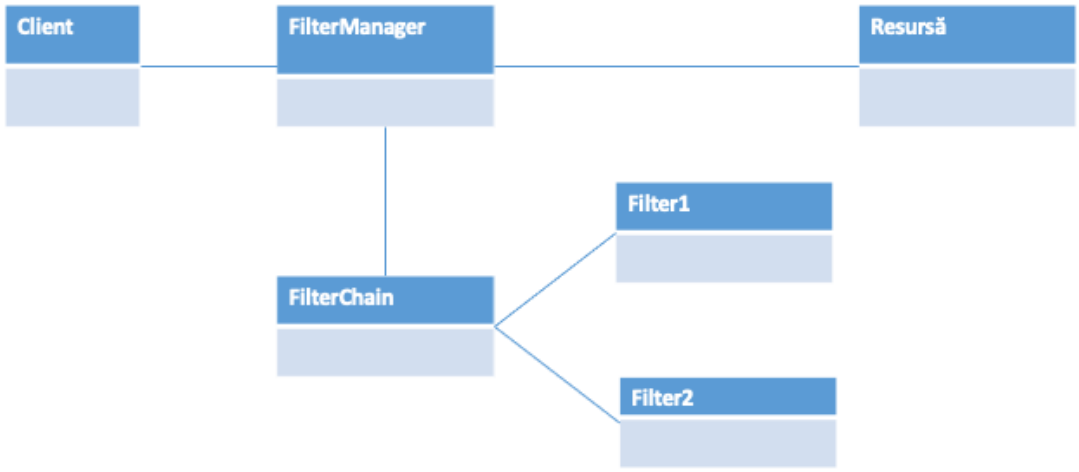
Intercepting Filter

- Anumite cereri necesită o serie de prelucrări în vederea utilizării ulterioare a acestora
  - Modificări
  - Verificări
  - Decompresie
- Operații
  - Pre-prelucrare
  - Post-prelucrare
- Se aplică cererilor și răspunsurilor

Implementare

- Definirea de filtre bazate pe o interfață comună
- Filtrele se activează automat la
  - Apariția unei cereri
  - Furnizarea unui răspuns

Diagrama de clase



Componente

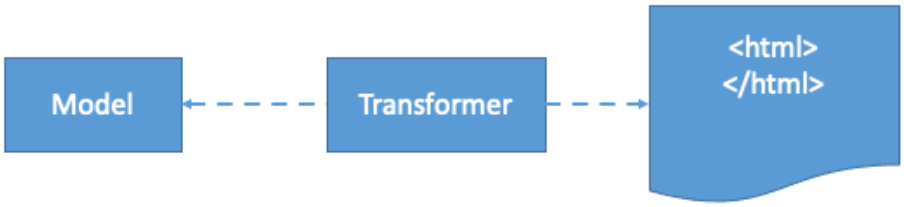
- **FilterManager**
  - Gestionează prelucrările filtrelor
  - Creează **FilterChain** cu filtrele corespunzătoare, în ordinea stabilită
  - Inițiază prelucrările
- **FilterChain**
  - Colecție ordonată de filtre independente
- **Filter1, Filter2**
  - Filtre asociate **Resursei**
  - Coordonate de **FilterChain**
- **Resursa**
  - Resursa solicitată de client

Transform View

Transform View

- Procesează un element al domeniului (model) și îl transformă în format HTML
- Intrare: date
- Ieșire: HTML

Diagrama



Componente

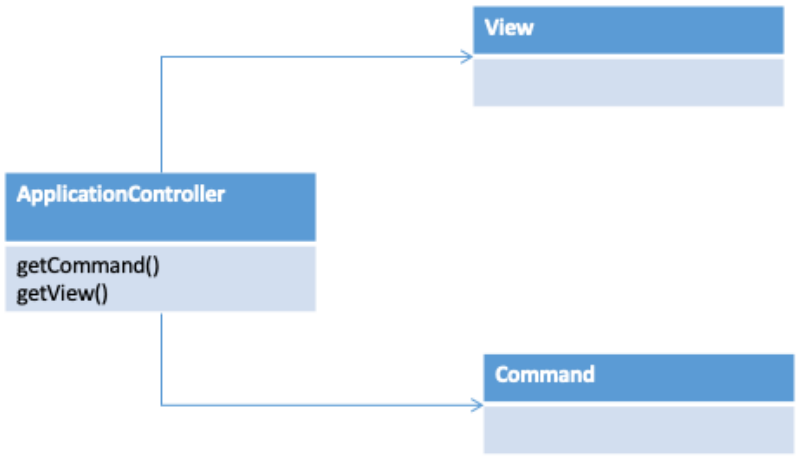
- Obiectele modelului care trebuie afișate într-o pagină Web
- Componenta care efectuează transformarea
- Pagina HTML asociată rezultatului

Application Controller

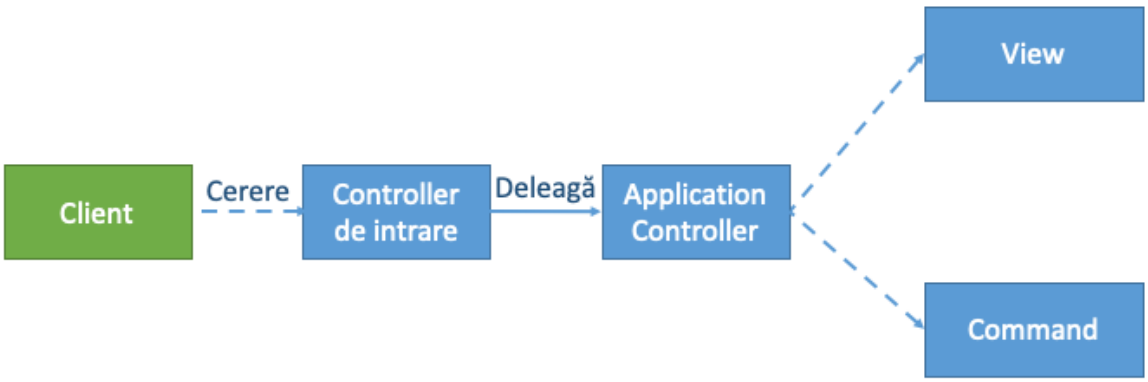
Application Controller

- Componentă centralizată pentru gestiunea:
  - Navigației în cadrul interfeței
  - Fluxului aplicației
- Degrevează controllerul de intrare de anumite activități
- Responsabilități:
  - Decide ce comandă se execută
  - Decide ce view se afișează

Diagrama de clase



Diagrama

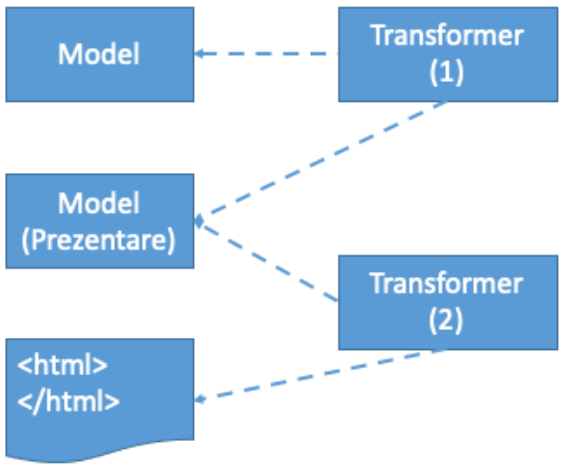


Two Step View

Two Step View

- Procesează un element al domeniului (model) și îl transformă în format HTML în două etape
- Transformarea este împărțită în două etape:
  - Datele modelului -> Logica prezentării (fără formatare)
  - Logica prezentării (fără formatare) -> HTML
- Suport pentru diferite platforme

Diagrama



Componente

- Obiect asociat modelului care trebuie afișate într-o pagină Web
- Componenta care efectuează prima transformare
- Rezultat 1
  - Obiect de interfață, fără formatare
- Componenta care efectuează transformarea finală
- Rezultat final
  - Pagina HTML asociată obiectului care trebuie reprezentat

Referințe

- .NET Design Patterns, http://www.dofactory.com/net/design-patterns
- Data & Object Factory, *Gang of Four Software Design Patterns*, Companion document to Design Pattern Framework™ 4.5, 2017
- Data & Object Factory, *Patterns in Action 4.5*, A pattern reference application, Companion document to Design Pattern Framework™ 4.5, 2017
- Design Patterns | Object Oriented Design, http://www.oodesign.com/
- Design patterns implemented in Java, http://java-design-patterns.com/patterns/
- R. Fadatare, *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall, 2013
- M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford, *Patterns of Enterprise Application Architecture*, Addison Wesley, 2002
- E. Freeman ș.a, *Head First Design Patterns*, O'Reilly, 2004
- J.D. Meier et al, Application Patterns, http://apparch.codeplex.com/wikipage?title =Application%20Patterns, 2009
- M. Richards, Software Architecture Patterns, O'Reilly, 2015
- D. Schmidt, M. Stal, H. Rohnert and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, Volume 2, John Wiley & Sons, 2000
- A. Shivets, *Design Patterns Made Simple*, http://sourcemaking.com
- Stack Overflow, https://stackoverflow.com/
- D. Trowbridge et. al, *Enterprise Solution Patterns Using Microsoft .NET*, Version 2.0, Microsoft, 2003