Software Engineering of Internet Applications

Section 3: Lecture 1

Enterprise Information System Patterns

Dr Laurence Dawson

laurence.dawson@kcl.ac.uk

contact@laurencedawson.com

15 February 2016

KING'S College LONDON

Course outline

- SIA split between myself and Dr Kevin Lano

- 4 weeks – Dr Kevin Lano
- 5 weeks – Dr Laurence Dawson
- 1 week – Dr Kevin Lano (revision lecture)

New sides are based upon the original slides produced by Dr Lano (*siaslides.pdf*)

Lecture structure

- EIS patterns
  - Lecture 1 (200 - 215)
  - Lecture 2 (216 - 227)
  - Lecture 3 (228 - 240)

- Web Services and EIS Technologies
  - Lecture 4 (251 - 261)
  - Lecture 5 (262 - 279)

Lecture slides & code examples

- The original slides, cut up slides and reworked slides will all be available Keats

- Code examples will be available on Github @

  - https://github.com/laurencedawson/6CCS3SIA
  - https://goo.gl/ycOEtU
  - Assuming everyone is familiar with git?

Worked examples with Netbeans

Download:

https://netbeans.org/downloads/index.html

Lecture outline

1) Introduction to EIS Patterns

- Presentation tier patterns
- Business tier patterns
- Integration tier

2) Presentation tier patterns

- *Intercepting filter* + worked example
- *Front controller + worked example*

1. Introduction to EIS Patterns

- One solution to complexity of EIS design is to provide patterns
  - *Otherwise known as standard solutions for EIS design problems*
  - Apply at different tiers (*see 5 tier architecture siaslides.pdf p70)*
  - Reusable

- Patterns define microarchitecture within an EIS
  - Each pattern used to implement a required property or function of a system

- The following slides will provide an overview to many different patterns across the presentation, business and integration tiers
  - * indicates a pattern later discussed in more detail

Recommended reading

J2EE Design Patterns

William Crawford, Jonathan Kaplan

http://shop.oreilly.com/product/9780596004279.do

Presentation tier patterns (6)

- **Intercepting filter***
  - Defines a structure of pluggable filters to add pre and post-processing of web requests/responses
    - e.g. security checking, XSS attack screening etc
- **Front controller***
  - Defines a single point of access for web system services, through which all requests pass
    - e.g. centralised handling of authentication etc
- These two patterns limit what a user can do
- Ensure a user works within expected bounds

Presentation tier patterns (cont.)

- **View helper**
  - Separates presentation and business logic by taking responsibility for visual presentation
    - Idea is to take out formatting code from business logic
    - Delegate formatting code out of business logic
    - **Ensures business logic separate from presentation logic**

- **Composite view***
  - Uses objects to compose a view out of parts (subviews)
    - Think oldschool php include
    - Subviews can be dynamic or static
    - Brought together to create a single template
    - Comparable to building an interface in Android out of Views

Presentation tier patterns (cont.)

Service to worker and dispatcher view patterns are very similar

- **Service to Worker**
  - This combines the *front controller* pattern (single point of access) and *view helper pattern (takes responsibility for visual presentation)*
  - Constructs presentation content in response to a request
- **Dispatcher View**
  - Similar to the *Service to Worker pattern* but defers content retrieval and error handling to the time of view processing

Business tier patterns (7)

- **Business Delegate**
  - *Think Java interfaces*
  - Provides an intermediary between presentation tier and business services
  - Reduces dependence of presentation tier on details of business service implementation
    - Doesn't contain business logic but knows how to locate and interact with business objects in the application

- **Value Object***
  - *Think Java Objects*
  - An object which contains attribute values of a business entity (entity bean), this object can be passed to presentation tier as a single item
    - Coalesces requests
    - Avoids cost of multiple getAttribute() calls on the entity bean
    - e.g. get customer name, age, address replaced with get customer info

Business tier patterns (cont.)

- **Session Façade***
  - Use a session bean as façade to hide complex interactions between business objects in one workflow / use case
    - Groups a set of activities into a session bean
    - Business tier object and hides other beans (session + entity) from the presentation tier
    - See P188 J2EE Design Patterns, William Crawford, Jonathan Kaplan
      - (available on Google Books)

- **Composite Entity**
  - Use an entity bean to represent and manage a group of interrelated persistent objects
  - The bean will collect information from multiple tables / objects into a single entity bean
    - e.g. Combine a patients medical records with a patients addresses
  - Avoids costs of representing group elements in individual fine-grained entity beans

Business tier patterns (cont.)

- **Value Object Assembler**
  - Builds a model using possibly several value objects from various business objects
  - The result is a composite value object that represents data from various business components
    - e.g. entity beans, session beans etc
- **Value List Handler**
  - Provide efficient interface to examine a list of value objects
    - e.g. Result of a database search
- **Service Locator**
  - Abstracts details of service/resource lookup, bean creation, etc.
  - Can be used by the Value Object Assembler to construct models

Integration tier patterns (2)

- **Data Access Object**
  - Provides abstraction of persistent data source access
    - Allows underlying datasource to change
    - Hides specifics of underlying datasource

- **Service Activator**
  - Implements asynchronous processing of business service components.

Patterns recap

- Many of the patterns are self explanatory
  - e.g. composite view
- You might already have used / come across certain patterns without realising
- The patterns are not specific to Java and can be used with other EIS application platforms

2. Presentation tier patterns

- **Intercepting filter**
  - Defines a structure of pluggable filters to add pre and post-processing of web requests/responses

- **Front controller**
  - Defines a single point of access for web system services, through which all requests pass

Intercepting filter

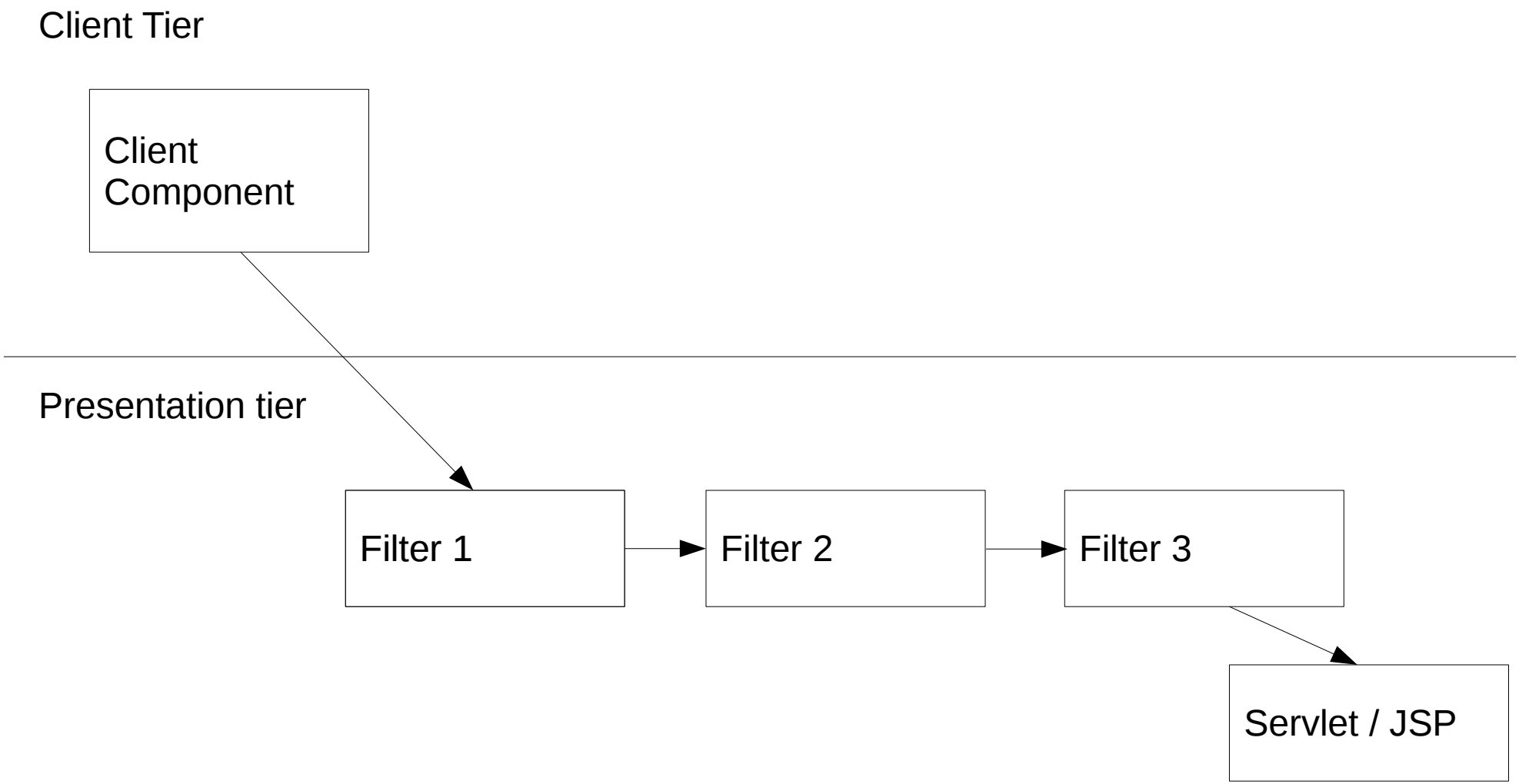
- When a client request enters a web application, it may need to be checked before being processed
  - Is the client's IP address from a trusted network
  - Does the client have a valid session?
  - Is the client's browser supported by the application?

Intercepting filter

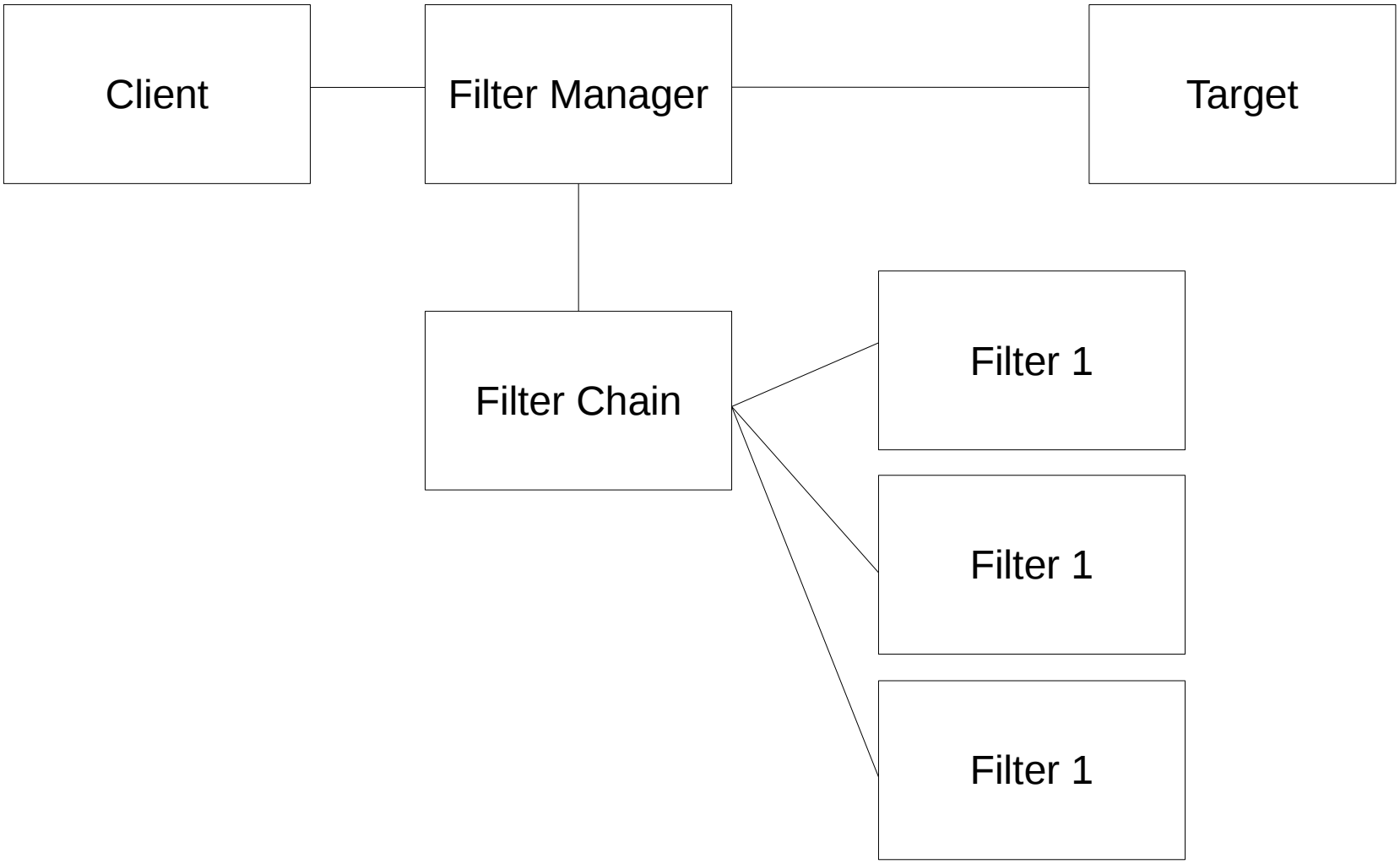
- Possible to code these as nested if tests, but is more flexible to use separate objects in a chain to carry out successive tests

Pattern elements

- **Filter manager**
  - Sets up filter chain with filters in correct order and initiates processing
- **Filter One, Filter Two ... Filter N**
  - Individual filters, which each carry out a single pre/post processing task
- **Target**
  - The main application entry point for the resource requested by the client. It is the end of the filter chain



Intercepting filter architecture



Class Diagram

Worked example: Markdown processor

A user form accepts markdown in a form and upon submission renders HTML in the next page.

Filter the input markdown text:

1) Filter key words
2) Only allow Chromium based browsers (Chrome, Chromium)
3) Remove simple XSS attacks

Example available at:

https://github.com/laurencedawson/6CCS3SIA

2. Presentation tier patterns

- **Intercepting filter**
  - Defines a structure of pluggable filters to add pre and post-processing of web requests/responses

- **Front controller**
  - Defines a single point of access for web system services, through which all requests pass

Front controller

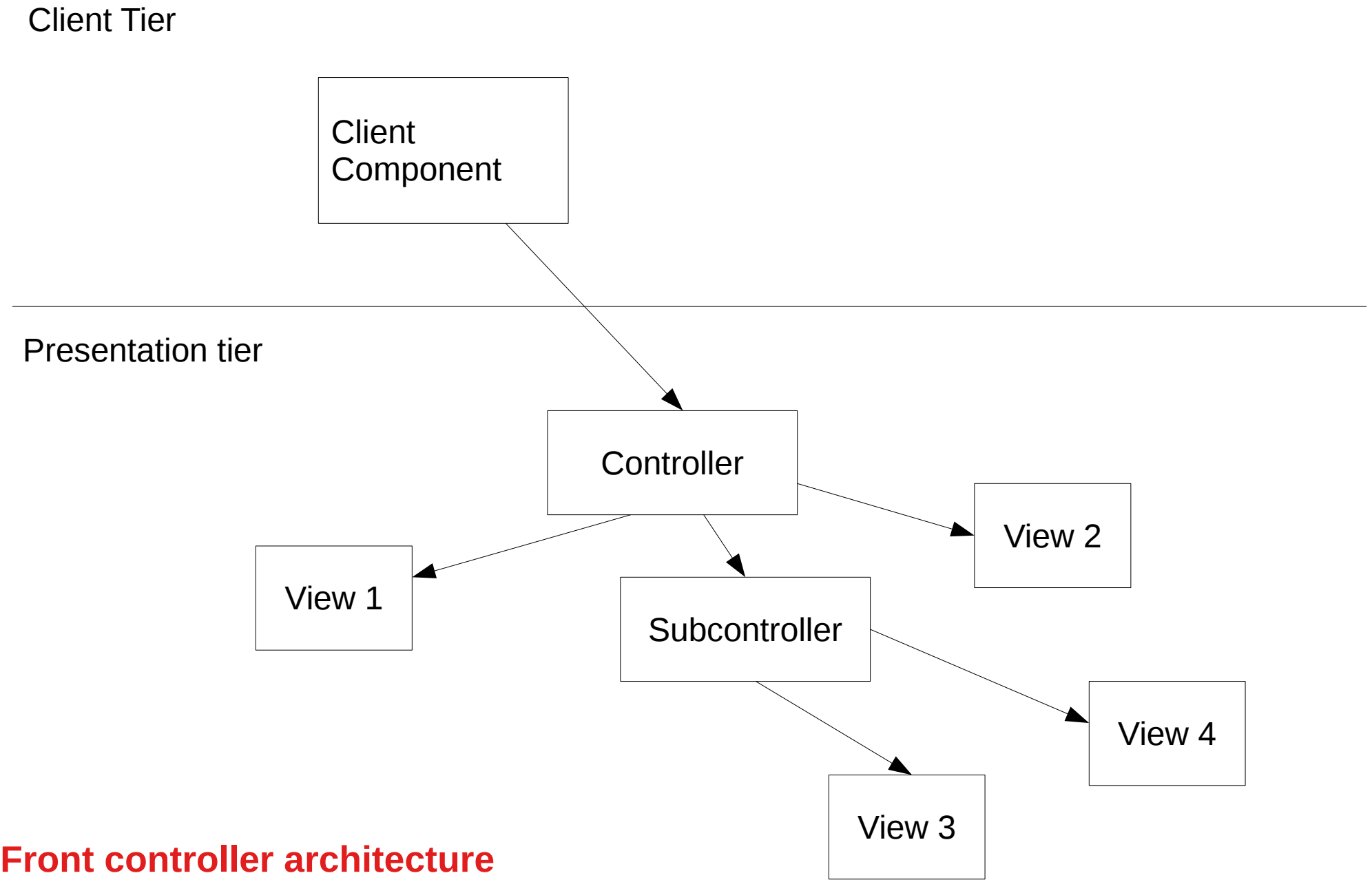
- This presentation tier pattern has the purpose to provide a central entry point for an application that controls and manages web request handling

- The controller component can control navigation and dispatching
- The pattern factors out similar request processing code that is duplicated in many views
  - e.g. Authentication checks in several pages

- It makes it easier to impose consistent security, data, etc, checks on requests.

Pattern elements

- **Controller**
  - Initial point for handing all requests to the system
  - Forwards requests to subcontrollers and views
- **Subcontroller**
  - Responsible for handling a certain set of requests
    - e.g. All those concerning entities in a particular subsystem of the application
- **View 1, View 2 ... View N**
  - Components which process specific requests, forwarded to them by the controller



Front controller architecture

Worked example: Post submission

An open user form requires a user to be logged in to submit text

1) If a user is logged in, submit the text
2) If a user isn't logged in, redirect them to a registration page
3) If the page type isn't handled, redirect them elsewhere

Example available at:

https://github.com/laurencedawson/6CCS3SIA

References

J2EE Design Patterns

O'reilly

http://archive.oreilly.com/pub/a/onjava/2002/01/16/patterns.html?page=2

Software Engineering of Internet Applications

Dr Kevin Lano

http://www.dcs.kcl.ac.uk/staff/kcl/6ccs3sia/siaslides.pdf