University of Glasgow

Web Application Frameworks

Web Application Development 2

Lecture Outline

- Model View Controller
  - An Architectural Design Pattern
- What is a Web Application Framework
  - Definition
  - Characteristics
  - Benefits and limitations
- Some common frameworks

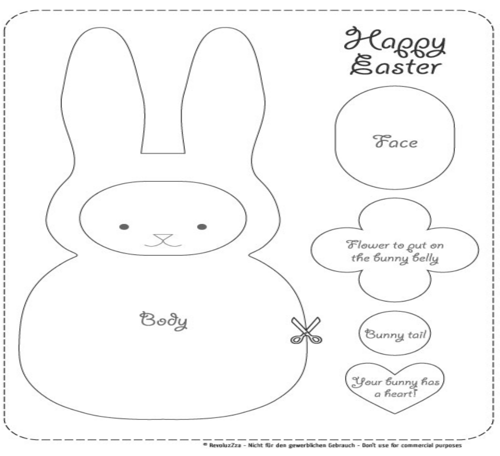
Re-inventing the Web App

- After developing several web applications (from scratch), it rapidly becomes clear that:
  - There is lots of coding overhead and 'boiler plate' code
  - Typically the same tasks are repeated over and over again
    - e.g., access a database, process then present results in HTML
  - There is a need for separation of concerns
    - Distribution of the main components / interactions to maximize code re-use, provide robustness, aid in debugging, enable scalability, etc

Pre-Fabricated Wheels

- Web frameworks typically provide (some of) the following features:
  - User authentication, authorisation, security
  - Database abstraction (or Object-Relational Mapping)
  - Template system
  - AJAX sub-framework
  - Session Management
  - An Architecture usually based on **Model-View-Controller**

An Underlying Pattern



Model-View-Controller

What is a Design Pattern?

- Serve as a tool to communicate ideas, solutions, and knowledge about commonly recurring design problems
- User interface design patterns help designers and developers create the most effective and usable interface for a particular situation
- Thus, each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution
- Patterns can be expressed hierarchically, with each layer representing a different level of granularity, and there may be many different ways to (physically) implement each pattern

Problem dealing with GUIs

- To separate the concerns between:
  - Presentation Logic
  - Business/Application Logic
  - Data Model
- The architecture pattern **Model View Controller** was developed
- This maps the traditional input, processing, output roles into the Graphical User Interface realm

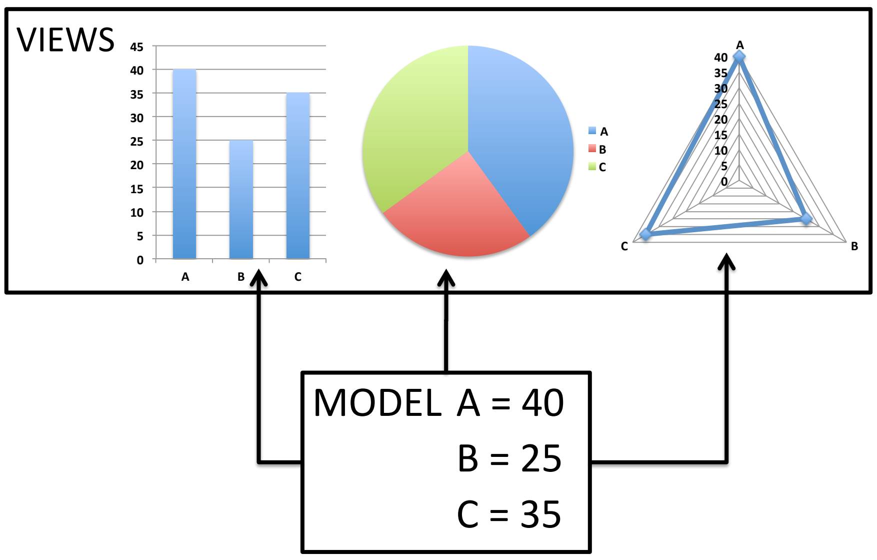
Model View Controller

- Invented in the 1970s at Xerox Parc
- The **controller** interprets mouse and keyboard inputs and maps these to actions
- These commands are sent to the **model** or **view** to enact the appropriate change
- The **model** manages the data elements
  - responding to queries about its state,
  - and updating its state
- The **view** manages the display for presenting the data

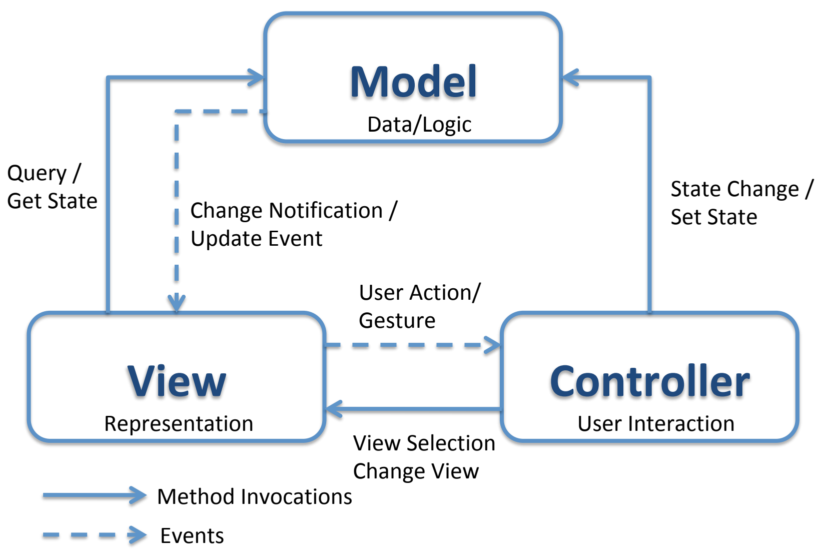
Model View Controller

- The model, this layer contains code that operates on the application data.
  - Any actions wanted to be executed on the raw data must go through this layer. Definitions of how the application work with data (commonly CRUD: create, read, update, or delete) are written here.
- The view, this is the presentation layer.
  - It defines how your pages should look like to the user, how the application presents data, or how a user can submit certain instructions to be executed by the application.
- The controller, this component acts as the orchestrator of the application.
  - It controls the flow of the program. It receives user commands, processes them, and then contacts the model, and finally instructs the view to display appropriately to the user.

Model and Views



MVC Architecture



Model

- Models represent application data and the domain logic
- Notifies views when it changes and enables the view to query the model
- Allows the controller to access application data functionality encapsulated by the model

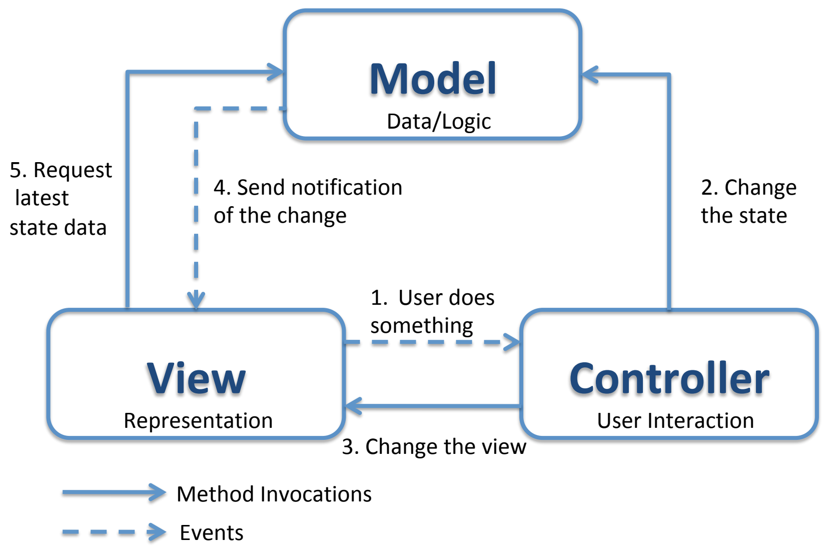
View

- A view is a (visual) representation of its model and acts as a presentation filter.
  - i.e., it renders the contents of the model
  - Specifies how the model data should be presented
- A view is attached to its model (or model part) and gets the data necessary for the presentation from the model by asking questions
  - So when the model changes, the view must update its presentation
    - Push Model: the view registers itself with the model for change notifications
    - Pull Model: the view is responsible for calling the model when it needs to retrieve the most current data
- The view is responsible for forwarding user requests / gestures to the controller

Controller

- Defines the application behaviour
- A controller is the link between a user and the system
- Interprets user requests / gestures and maps them into actions
  - for the model to perform and
  - arranges for relevant views to present themselves in appropriate places on the screen

MVC Architecture



Model View Controller

Advantages

- Enable independent development & testing
- Easier to maintain
- Provides reusable views & models
- Synchronized views and multiple simultaneous views
- Helps enforce logical separation of concerns

Disadvantages

- Some initial overheads splitting up concerns
  - Increased overheads in development (i.e. 3 classes vs 1)
- Especially for very simple applications
- Debugging can sometimes be a problem
- Requires the developers to understand patterns

Model View Controller in Django

- What about templates..?
- MVC in Django can be confusing, because its 'views' receive input, query & process data, and might be considered part of the controller role in traditional MVC
- Often say **MVCT** or just **MTV** rather than **MVC**:
  - **Models** describe your database
  - **Controller** is handled by
    - the Django Framework
    - URL parser maps urls to views, where processing may occur
  - **Templates** describe how the data is presented

Frameworks

- As with real world frameworks (e.g. building frame and vehicle chassis), software frameworks provide design and partial implementation for a particular domain of applications
- Frameworks allow developers to create applications more efficiently by providing **default functionality**, whilst allowing them to **extend and override** to suit their specific purposes

Classic Framework Definitions

- There are several interpretations of a framework:
  - A framework is a set of classes that embodies an **abstract design** for solutions to a **family of problems**
  - A framework is a set of **prefabricated software building blocks** that programmers can **use, extend, or customize** for specific computing solutions;
  - Frameworks are large **abstract applications** in a **particular domain** that can be **tailored** for individual applications
  - A framework is a **reusable software architecture** comprising both **design and code**

Why do we need frameworks?

- Virtually all web applications have a common set of basic requirements
  – (user management, security, password recovery, sessions management, database management, etc)
- Frameworks encapsulate thousands of hours of experience, knowledge and know-how
  – Improved over each iteration, debugged, secured, etc
- Often can handle reasonably high loads and traffic out of the box

Framework Characteristics

- **Inversion of Control**
  – Framework is responsible for the application control flow
- **Default Behaviour**
  – Framework must provide some 'useful' functionality related to the application domain
- **Extensibility**
  – Hot-spots designed to be extended
  – Allow developer to customize their application specifically for a particular purpose
- **Non-modifiable Framework Code**
  – Key components of the framework cannot be altered
  – Not strictly non-modifiable, but typically just used, though contributions back to the framework are often subject to the framework creators or open source community

Framework Characteristics

- **Advantages:**
  - – Enables rapid development
  - – Concentrate on unique application logic
  - – Reduces boiler plate code
  - – Already built and tested, increased reliability
  - – Increased security (generally)
  - – High level of support for basic common functionality

Framework Characteristics

- **Disadvantages:**
  - – Impose a certain model of development (80% easy / 20% hard)
  - – Frameworks can introduce code bloat
  - – Levels of abstraction generally introduce performance penalties
  - – Difficult to overcome the steep learning curve
  - – Can be poorly documented
  - – A bug or security risk in the framework can seriously compromise the application

Framework vs Libraries?

- Framework is about reusing behaviours by controlling how abstract classes and components interact with each other
  - A framework calls your application code
- A library is a collection of classes which provide reusable functionality
  - Your application code calls the library

Some Examples of Web Application Frameworks

Many Web App Frameworks

- **JavaScript**
  - Angular, Backbone, Ember, React, Vue, etc.
- **Java**
  - Spring, Struts, Grails, Google Web Toolkit, etc
- **PHP**
  - Symfony, Cake, CodeIgniter, Laravel, etc
- **Python**
  - Django, FastAPI, Bottle, Flask, TurboGears, Pyramid, Zope, etc
- **Ruby**
  - Rails, Camping, Merb, Sinatra, Padrino, etc
- ASP.NET, ColdFusion, C++, Tcl, Ocaml, Scala, Groovy, etc

http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

Common Framework Functionality

- **Web Template System**
  - to provide pre-defined pages that load dynamic content
- **Caching**
  - to reduce perceived lag
- **Security**
  - to provide authentication and authorization functionality
- **Database access and mapping**
  - to speed up working with databases and avoid using SQL

Common Framework Functionality

- **URL Mapping**
  - To enable handling of URLs and friendlier URLS
- **AJAX handlers and handling**
  - To create more dynamic pages that are more responsive
- **Automatic configuration**
  - To decrease the setup hassles, usually uses introspection and/or following conventions
- **Form Management**
  - To speed up the creation of forms and handling of forms

Why Use WAFS?

Why use WAFs?

- To enable rapid development that matches the rapid release cycle of the web
- To reduce the development effort of programming in n-different languages/technologies
  - Database access (possibly Object-Relational Mapping)
  - Templating HTML
- To manage the complexity of the increasingly large and sophisticated web applications by including library support for:
  - User Authentication
  - Session Management
  - Creating a Web Service

Why use WAFs?

- To reduce ‘boiler plate’ code in web applications
  - Particularly access and manipulation of DB
  - Often referred to as CRUD operations
  - Session management across multiple pages
- Web-apps have matured to a point where software engineering practices (including design patterns and frameworks) are becoming:
  - increasingly useful
  - necessary
  - the norm

WAF Caveats

- They require an investment in learning the framework
  - **Learning vs. Building Trade-off**
- Sacrifice some flexibility for rapid development
  - **Flexibility vs. Efficiency Trade-off**
- Like client-side libraries, knowledge of one framework does not necessarily transfer to another
- Early stages of web framework eco-systems
  - There are many competing options at present
  - Eventually the most popular (few) will emerge

Summary

- **Model View Controller** Design Pattern
- **What is a Web Application Framework**
  - Definition
  - Characteristics
  - Benefits and limitations