Design Patterns

Higher Diploma in Science in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Web Presentation & Session Patterns

Enterprise Patterns

- Session State
  - Client Session
  - Server Session
  - Database Session
- Offline Concurrency
  - Optimistic Lock
  - Pessimistic Lock
  - Coarse Grained Lock
  - Implicit Lock
- Distribution
  - Remote Facade
  - Data Transfer Object
- O/R Metadata Mapping
  - Metadata Mapping
  - Query Object
  - Repository
- Base
  - Special Case
  - Money
  - Plugin
  - Mapper
  - Separated Interface
  - Service Stub
  - Gateway
  - Registry
  - Value Object
  - Record Set
  - Layer Supertype
- Web Presentation
  - View
    - Transform
    - Template
    - Two Step
  - Controller
    - Application
    - Page
    - Front
  - MVC
- Domain Logic
  - Transaction Script
  - Domain Model
  - Table Module
  - Service Layer
- Data Source
  - Gateway
    - Table Data
    - Row Data
  - Active Record
  - Data Mapper
- O/R Structure
  - Identity Field
  - Mapping
    - Foreign Key
    - Association
    - Dependent
    - Inherritance
  - Embedded Value
  - Serialized LOB
  - Inheritance
    - Single Table
    - Class Table
    - Concrete Table
- O/R Behaviour
  - Lazy Load
  - Identity Map
  - Unit of Work

Web Presentation Patterns

- Clear advantages:
    - no client software to install, a common UI approach, and easy universal access.
- However, intimate knowledge of HTTP now seen as important. Previous attempt to ‘abstract away’ HTTP to lower layers have incurred excessive complexity costs
- Modern web framework fully expose HTTP, and assume developers are comfortable with the primary mechanisms

Web Presentation Patterns

- Model View Controller
- Page Controller
- Front Controller
- Template View

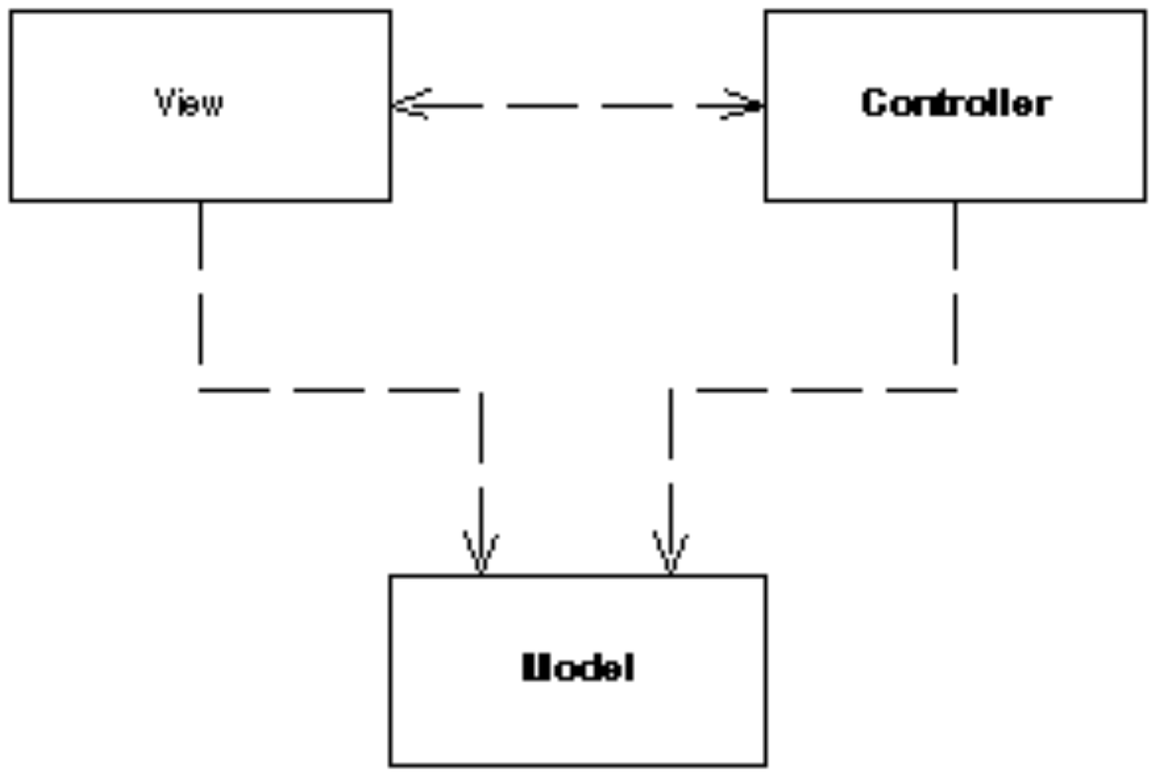
Mode View Controller

Splits user interface interaction into three distinct roles.

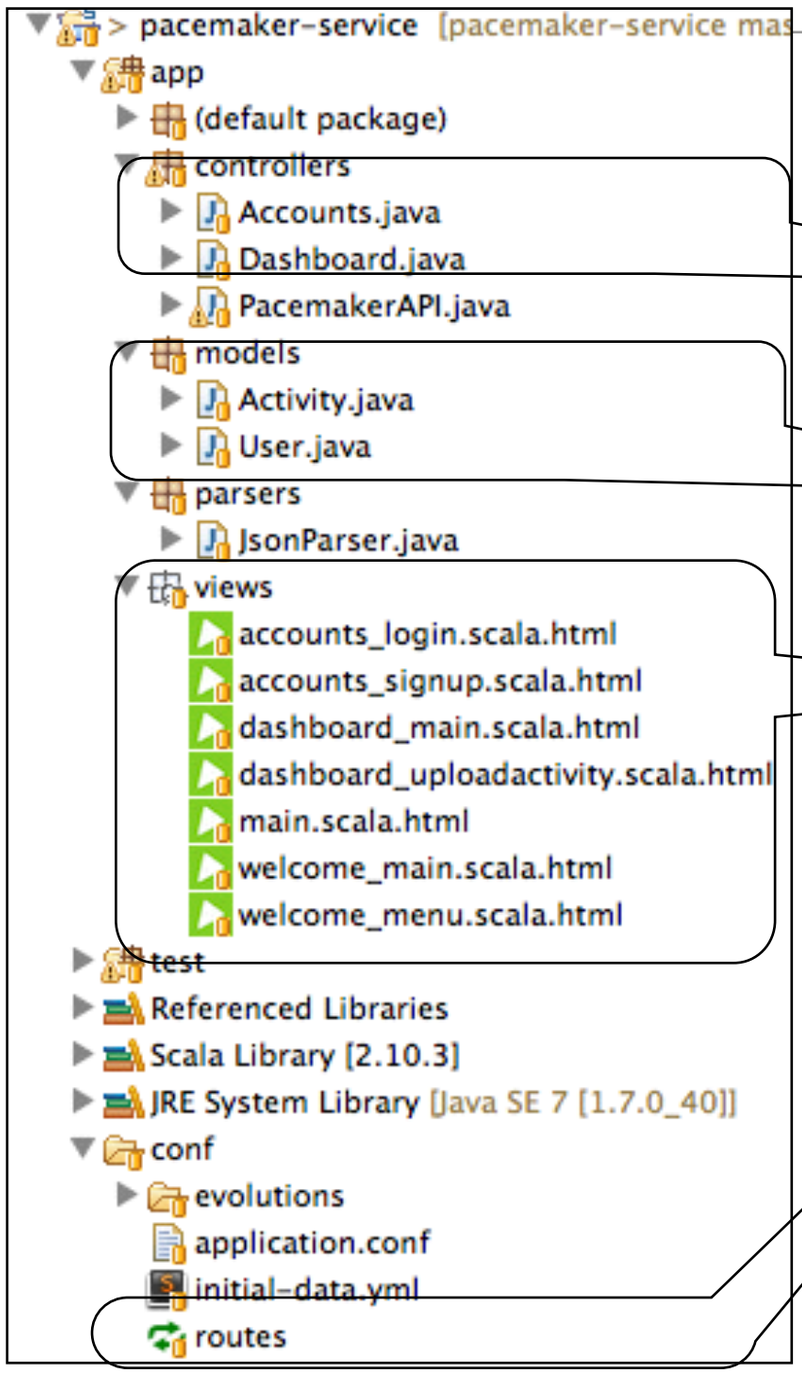
- Controller Model View Controller (MVC) is one of the most quoted (and most misquoted) patterns around.

- It started as a framework developed by Trygve Reenskaug for the Smalltalk platform in the late 1970s.

- Since then it has played an influential role in most UI frameworks and in the thinking about UI design.



MVC in pacemaker-service

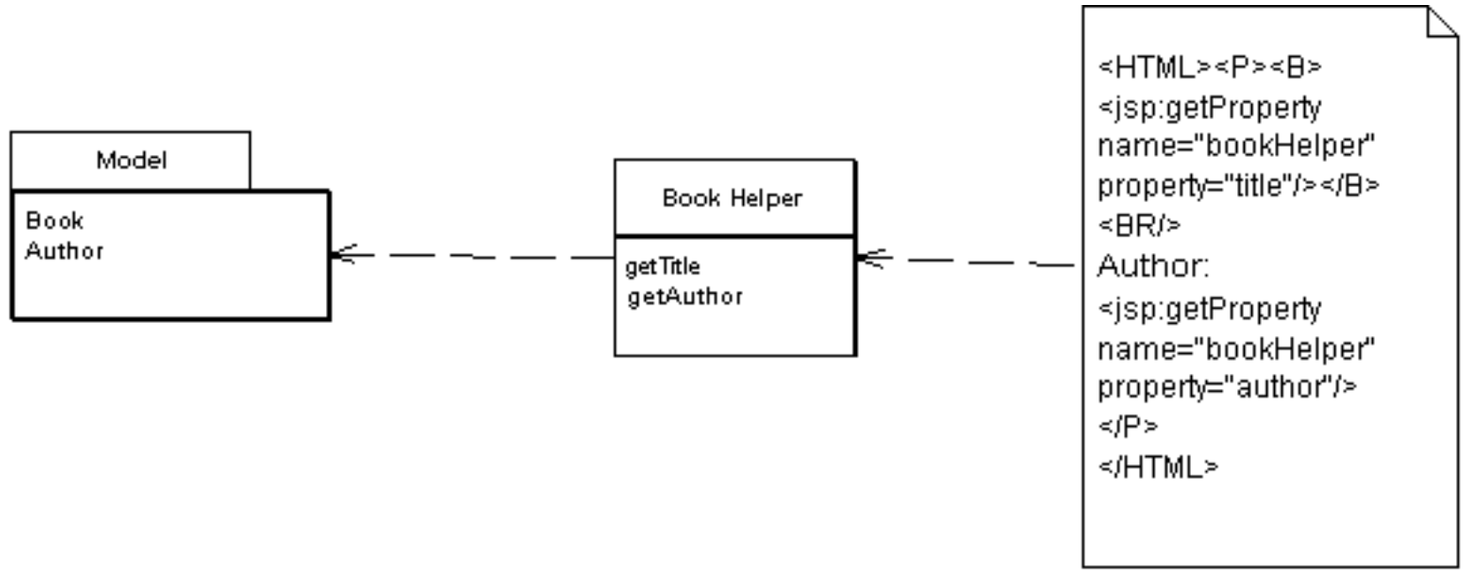


- Routes define acceptable URLs and map them to Actions
- Actions interact with Domain logic and Render..
- Views, which are served to the browser

Template View

Renders information into HTML by embedding markers in an HTML page.

- Compose a Dynamic Web page as you do a static page but
  - put in markers that can be resolved into calls to gather dynamic information.
- Since the static part of the page acts as a template for the particular response



pacemaker-service Template Method

```
public class Accounts extends Controller
{
  //...
  public static Result login()
  {
    return ok(accounts_login.render());
  }
}
```

- Templates in Play are compiled as scala functions
- Compile time check + potential efficiency benefits

```
@()

@main("Welcome to Pacemaker") {
  @welcome_menu()

 <section class="ui raised segment">
   <div class="ui grid">
     <aside class="ui six wide column">
      <img src="@routes.Assets.at("images/pacemaker.jpg")" class="ui medium image">
     </aside>
     <div class="ui ten wide column fluid form">
       <div class="ui stacked segment">
         <form action="/authenticate" method="POST">
           <h3 class="ui header">Log-in</h3>
           <div class="field">
             <label>Email</label>
             <input placeholder="Email" type="text" name="email">
           </div>
           <div class="field">
             <label>Password</label>
             <input type="password" name="password">
           </div>
           <button class="ui blue submit button">Login</button>
         </form>
       </div>
     </div>
   </div>
</section>
}
```

Sessions

- Client Session State
  - Stores session state on the client.
- Server Session State
  - Keeps the session state on a server system in a serialized form
- Database Session State
  - Stores session data as committed data in the database.

Play : Sessions and Flash Scopes

- If you have to keep data across multiple HTTP requests, you can save them in the Session or Flash scopes.
    - Data stored in the Session are available during the whole user Session,
    - Data stored in the Flash scope are available to the next request only.
- Session and Flash data are not stored by the server but are added to each subsequent HTTP request, using the cookie mechanism.
    - This means that the data size is very limited (up to 4 KB) and that you can only store string values.
    - Cookie values are signed with a secret key so the client can’t modify the cookie data.
- The Session is not intended to be used as a cache. If you need to cache some data related to a specific Session, you can use the Play built-in cache mechanism and use store a unique ID in the user Session to keep them related to a specific user.

Session Object Encapsulates Session mechanisms

- Use email as application specific session id
- On each request, retrieve this id to determine current user details

```
public class Accounts extends Controller
{

  public static Result logout()
  {
    session().clear();
    return ok(welcome_main.render());
  }

  public static Result authenticate()
  {
    Form<User> boundForm = loginForm.bindFromRequest();
    if(loginForm.hasErrors())
    {
      return badRequest(accounts_login.render());
    }
    else
    {
      session("email", boundForm.get().email);
      return redirect(routes.Dashboard.index());
    }
  }
}
```

```
public class Dashboard extends Controller
{
  public static Result index()
  {
    String email = session().get("email");
    User user = User.findByEmail(email);
    return ok(dashboard_main.render(user.activities));
  }
}
```

application.conf

```
# Secret key
# ~~~~~
# The secret key is used to secure cryptographics functions.
# If you deploy your application to several instances be sure to use the same key!
application.secret=":qEJLP]R2D8prCCf9`@F4d1q_`URxLT3CmxucR7ued`rfspew?X?S_J;P;`VsZ^R"
```