Architectural Patterns and Styles - Notes

Priscilla Madigoe

08 March 2016

1 Architectural Patterns

1.1 Introduction

Often times source code needs to be organised so that clear roles, responsibilities and relationships of different modules of the code can be well defined. Architectural patterns help in making this possible and numerous types will be discussed in this section. Using these patterns, code becomes easier to maintain, manage and visualise. They also make understanding of how each component works in a system easier. They are reusable solutions to a commonly occuring problems within a given context in Software Architecture.

1.2 Types of Architectural Patterns

- Layered Pattern
- Client-server
- Microkernel
- Master-slave
- Pipe-filter
- Hierarchical
- Model View Controller
- Blackboard Pattern

1.3 Architectural Patterns Proposed for PAPERS

1.3.1 Model View Controller

- Overview

  In the Model-View-Controller Pattern, an interactive application is divided into three parts: the Model is an object representing and storing the data and activities, the View displays information to the user and the Controller offers a way to change the state of the Model. To implement this patten for the system, the Python Django Framework will be used.

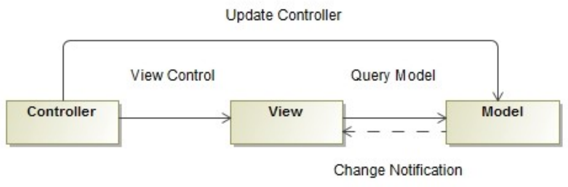

Figure 1: Model View Controller Pattern

- Motivation
  Separation of responsibilities is useful because functions can work normally when certain subsystems are changed. The Model View Controller makes this a possibility and hence it is chosen for use in this context.

- Components of the Model View Controller Pattern

  1. Model
     The Django Framework provides a Model functionality that handles the creation and storage of data in a relational databse that will contain essential fields and behaviours of the data stored for the system.

  2. View
     Normally, the View would be the HTML page or the Android Graphic User Interface, or GUI that displays data from the relational databse to the user for interaction purposes. However,

Django implements the View as a ”Python callback function for a particular URL, because that callback function describes which data is presented” to the user. Thus the View would be both the User Interface and Django’s implementation of the View for the system.

3. Controller

   The Django Framework itself will be the Controller as it will provide appropriate functionality for data manipulation. It will be used along with the REST, or Representational State Transfer API to retrieve data from the Android and the Web Interface.

1.3.2 Layered Pattern

- Overview

  The Layered Pattern assists in structuring applications that can be decomposed into subtasks. Each subtask provides services to the layer above it and services in that layer are implemented using those from the layer below it.

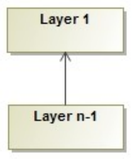

Figure 2: Layered Pattern

- Motivation

  A system typically consists of subsystems that have a certain functionality. These functions need to be placed in layers so that their services can be used by other layers and a change in behaviour of one layer does not affect the layers below it. This assists in developing different layers independently.

- Components of the Layered Pattern
  This pattern will be used to separate the internal functions within each subsystem and each function will be placed in a layer. It will be used to structure the Model View Controller Pattern described above to provide additional decoupling and structure to the system.

1.3.3 Master-Slave

- Overview
  The Master-slave Pattern can be described as a model where one process (Master) has unidirectional control over one or more processes that are known as Slaves.

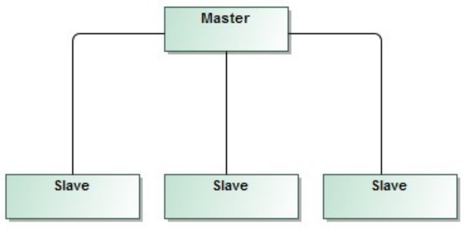

Figure 3: Master Slave

- Motivation
  An interactive application typically consist of multiple users that use it at different rates. These users make requests that usually expect a response to either use the data or edit it in some way and thus is imperative for the application have an ability to collect user information simultaneously.

- Components of the Master-Slave Pattern
  This pattern will be used to make the system concurrent, i.e to enable multiple users to use it simultaneously and continuously at different rates.

  - Master
    The Master will be the server that will respond to each request

  it receives from the Slave using the system, and will also provide instructions to these Slaves.

  - Slaves
    The Slaves will be the clients connecting to the server and will send requests concurrently to achieve a certain goal.

1.4 Other Patterns

1.4.1 Client Server Pattern

- Client Application provides access to humans, or systems (Fritz, 2012)
- Requests are typically handled in separate threads on the server.
- Inter-process communication causes overhead.
- Requests and data have to be transformed because they have different representation in Server and Client.

1.4.2 Microkernel

- Used for applications that must adapt to changing system requirements.
- This pattern provides minimal functionality (Internal Servers) as a building block and customizable components that can be adapted accordingly (External Servers).
- It provides an Adapter that provides a single access point (Fritz, 2012)
- It offers a more complex process management if not provided by architecture (Fritz, 2012)

1.4.3 Pipes and Filters

- Consists of Pipes and Filters:
  - Pipe - Input and Output
  - Filters - Stateless sevices or pure functions.
- Assemble higher-level stateless services from lower level ones (Fritz, 2012)
- It is not used because it is difficult to use for interactive processes such as Web Applications.

1.4.4 Hierarchical

- It is a recursive containment hierarchy (Fritz, 2012)
- It is not used due to its lack of flexibility.

1.4.5 Blackboard Pattern

- Blackboard repository hosts problem specification and current state of solution (Fritz, 2012)
- It is typically used where deterministic solutions are unavailable.
- It is not used because it requires complex synchronization and access control techniques.