CS 3354 Assignment 4

Daniel Crawford

09/25/2018

1.

1. 1. MVC

Summary:

This pattern separates presentation and interaction from the system data, which is made up of the model component, view component and the controller. The model component manages the system data and its operations. View component defines and manages how data is presented to the user. Last, the controller manages user interaction.

Strengths:

Data can change independently of its representation

Supports presentation of the same data in different ways with the changes made

in one representation shown in all of them.

Weaknesses:

Could require additional code which impacts complexity when data model and

Interactions are simple.

* 1. Layered

Summary:

Organizes system into layers with relationships in functionality each layer. Each layer will provide a service to the layer above it. Lowest level represents core services.

Strengths:

Entire layers can be replaced, but interface must be maintained.

Redundant facilities can be provided in layers to improve dependability.

Weaknesses:

Creating a separation between layers is difficult, so a high-level layer may need

to interact with a lower level layer.

Performance can take a hit due to multiple levels of interpretation.

* 1. Repository

Summary:

All data is managed in a central repository which is accessible to all system components. Components interact through the repository.

Strengths:

Components can have independence. No knowledge of other components is needed inside components.

Changes made in components can be propagated to all components.

Weaknesses:

Repository maintains everything, so if it fails then everything around it will

also fail.

* 1. Client & Server

Summary:

Functionality organized into services, with each service delivered from a separate server. Clients are users of these servers.

Strengths:

Servers can be distributed across a network.

General functionality can be available to all clients

Weaknesses:

Each server is a single point of failure, susceptible to DOS attacks

Performance unpredictable since it relies on a network.

* 1. Pipe and Filter

Summary:

Processing of data within a system organized in a way so that the processing component, or the filter, only carries out one type of data transformation. The pipe is the data flow that goes from component to another for processing.

Strengths:

Easy to understand and supports transformation reuse.

Workflow style matches structure of business processes.

Evolution through transformations is straightforward, can be implemented

as sequential or concurrent.

Weaknesses:

Format of data transfer agreed upon from communicating transformation All

transformations must parse input and put back into an agreed form. This will

increase overhead.

1. I do not believe that one of the architectures will specifically be better than the other. Each of these obviously hold their strengths in different areas. However, if I had to choose one of these architectures, I would personally choose the Repository architecture because I feel while it holds a single point of failure, it makes the rest of the system modular. In this case, if one part of the system components is broken, it can be easily fixed without worry of affecting other programs. A strong code base for the repository is absolutely needed, but afterwards components can be added swiftly.
2. MVC because the diagrams can send a message about what the product is trying to accomplish, but it does not help the implementation. Because the implementation is what the end goal of all software projects are, an architecture patterns needs to guide implementation better. If an architecture leads to more complex code, it really loses its use around the implementation when it could be most useful.

2.

1. Our project is an emergency/general purpose Traffic Light System that reports accidents and handles timing on traffic lights better. I believe that an MVC pattern is the correct architecture for this problem. The model component will be the point where all the data is handled from the cars. This server can decide the timing on traffic lights and recognize the accident from the data. The controller will be inputs by recognizing cars waiting at a traffic light which will go to the controller, which goes to the data to comprehend. After all this is done, the view component will have two sectors to it. The easy one is the one that is already defined for traffic lights: green, yellow blue. There also exists right turn green arrows and left, so that must also be considered. But a really important part to the usefulness of the system is the user interface for the accidents. This needs to be precise at to not alert police/ambulance/firetrucks to an accident that did not happen. Optimistically, it can provide a camera that shows the area so that the departments can decide themselves if they need to act. However, cheaper options may be needed and need to find a way how to visualize this in the view component.
2. There were three different patterns recommended for this system: Layered, Client and Server, and MVC. For MVC, it fits very well because the model can be what is handling all the data and deciding all the output. The controller is the input from the users simply congregating around the traffic light, so the input is detecting these people. View is the way the people will be able to view the camera, what the traffic light does, and it should always from from Green → Yellow → Red → Green. For Client & Server, it would be easy to implement into the system. However, we decided later that having a single point of f2ailure would be dangerous for a traffic light system, so if a DOS attack were to happen it could shut down all traffic lights. Lastly, Layered would be also easy to implement and would allow for substitution of different layers. However, it would be hard to separate component.

To conclude, we all felt that MVC was the best of the three suggest options. The other two options had significant drawbacks that would not work for a system as necessary. The MVC handles multiple areas of input, which is a great fit for our project which can take multiple inputs to optimize the traffic light. It also allows the data to change independently of its representation. For these reasons, MVC is the best fit for our project.