Activity: Architecture Patterns and Styles

Importance

Understanding software architecture patterns can help software designers address particular quality of service concerns in their software design.

Learning Objectives

- To understand some common architectural patterns.
- To understand the implementation of the MVC architecture / design pattern.

Success Criteria

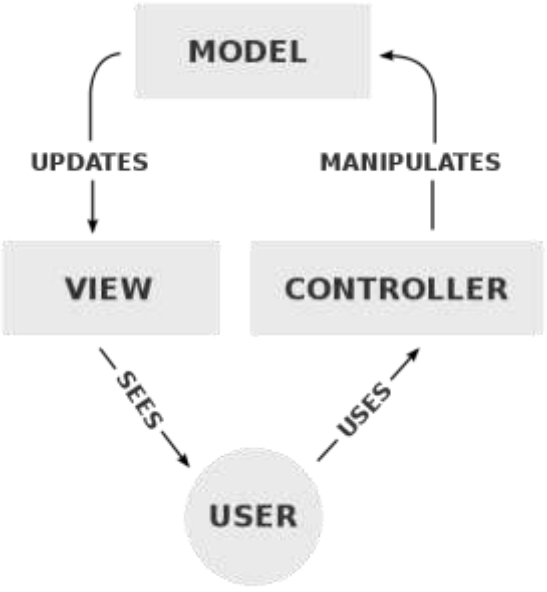
- To be able to define architectural patterns for a particular software system.
- To be able to write a small program that implements the MVC pattern and demonstrate its ability to easily support a change in the user display.

Resources

- Course notes on Software Architecture Patterns and Styles
- Online resources on the MVC pattern

Exercises

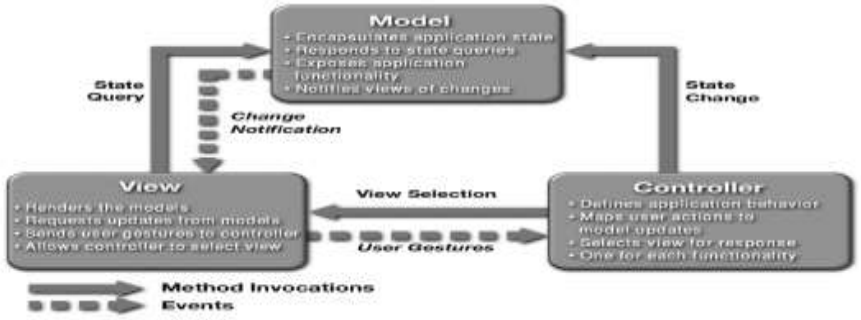
The MVC pattern has always been a misunderstood architectural pattern that is implemented in several different ways as well as there is not a clear understanding of the meaning of the interactions between the components. Fundamentally the pattern leverages a interaction pattern as shown below. The differences are primarily on how the update to the View is implemented. In



some examples the updates are direct calls to the View operators (Dependency from Model to

View) and on other examples the Observer pattern is implemented creating a stronger dependency from the View to the Model.

The MVC models in the course notes also show an optional dependency between the Controller and the View that supports the Controller changing views depending on User inputs.



1. For this exercise implement the MVC model on the provided Classes leveraging the Observer Java Class so that the *StoreView* is updated when customers are added to the *Store* through the *StoreController*. Submit the code and screen dumps of the test program showing that the Customers are printed out every time that they are updated.
2. Demonstrate the flexibility of the pattern by adding a new graphical output rather than the text output. The only Class that should be updated in the *StoreView.*
3. Going back to the Cash Register Requirements one can define the following significant components: *CashRegister*, *Display*, *Keyboard*, *TicketPrinter*, *Scanner*, and *Database*. Leveraging these components create UML or Box and Line diagrams showing:
    a. A Client-Server architecture where the Display and Keyboard are Clients to the CashRegsiter.
    b. A Pub-Sub architecture coordinating interactions among the components
    c. A pipe and filter architecture capturing the components and flow commencing from scanning the items to displaying the item and price. Note: Use a Data flow diagram for this as it is better suited. See https://www.visual-paradigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/