MVC / MVP

Reid Holmes

[Image from: http://merroun.wordpress.com/2012/03/28/mvvm-mvp-and-mvc-software-patterns-against-3-layered-architecture/]
Background

- MVC started w/ Smalltalk-80
- Java UI frameworks & EJBs reignited interest
- Also prevalent in GWT and .NET development
MVC Motivation

- UI changes more frequently than business logic
  - e.g., layout changes (esp. in web applications)
- The same data is often displayed in different ways
  - e.g., table view vs chart view
  - The same business logic can drive both
- Designers and developers are different people
- Testing UI code is difficult and expensive
- Main Goal: Decouple models and views
  - Increase maintainability/testability of system
  - Permit new views to be developed
Model

- Contains application data
  - This is often persisted to a backing store
- Does not know how to present itself
- Is domain independent
- Are often Subjects in the Observer pattern
View

- Presents the model to the user
- Allows the user to manipulate the data
- Does not store data
- Is configurable to display different data
Controller

- Glues Model and View together
- Updates the view when the Model changes
- Updates the model when the user manipulates the view
- Houses the application logic
- Loose coupling between Model and others
- View tightly cohesive with its Controller
Abstract topology

1. The **Controller** <<changes>> updates the **Model** with state changes.
2. The **Controller** <<updates state>> retrieves state from the **Model**.
3. The **Model** <<retrieves state>> notifies the **View** of state changes.
4. The **View** <<notifies of state changes>> receives notifications from the **Model**.
Concrete topology

```java
Factory f = GWT.create(Factory.class);
ViewController c = new ViewController();
View v = f.createView(c);
```

[hton.xml maps Factory.class to the right type]
Interaction mechanism

- User interacts with the UI (View)
- UI (View) notifies controller of changes
- Controller handles notifications, processing them into actions that can be performed on the model
- Controller modifies the model as required
- If the model changes, it fires modification events
- The view responds to the modification events
Benefits and tradeoffs

Pro:
- Decouple view from model
  - Support multiple views [collaborative views]
  - Maintainability [add new views]
  - Split teams [relieve critical path]
  - Testability [reduce UI testing]

Con:
- Complexity [indirection, events]
- Efficiency [frequent updates, large models]
Compound Pattern

- MVC (and other similar patterns) rely upon several more basic design patterns

- In MVC:
  - View/Controller leverage the strategy pattern
  - View is often a composite pattern
  - View/Model interact through the observer pattern
  - Other meta-patterns rely upon similar lower-level design patterns
MVP Motivation

- Take MVC a tiny bit further:
  - Enhance testability
  - Further separate Designers from Developers
- Leveraged by both GWT and .NET
Model

- Contains application data
  - This is often persisted to a backing store
- Does not know how to present itself
- Is domain independent
- Often fires events to an Event Bus
View

- Thin UI front-end for controller
- Does not store data
- Can be interchanged easily
- Does not ever see or manipulate Model objects
- Only interacts with primitives
  - e.g., (setUser(String) instead of setUser(User))
Controller

- Glues Model and View together
- Updates the view when the Model changes
- Updates the model when the user manipulates the view
- Houses the application logic
MVP Topology

View

Presenter

Model

Event Bus

1. <<notifies>>

2. <<updates, retrieves state>>

3. <<notifies of state changes>>

4. <<refresh>>
Concrete MVP Topology

- MobileView
- BrowserView
- MockView

- OutlineView
- MockOutline

- ViewController
- OutlineController

- App Controller

- Model

<<notifies of state changes>>

Event Bus
Concrete Example

```java
Factory f = GWT.create(Factory.class);
AppController ac = new AppController(f);
ac.showMain();

View v = f.createView(new ViewController());
Outline o = f.createOutline(new OutlineController());

public interface IJoinTripView {
    Widget asWidget();

    public void setPresenter(Presenter presenter);

    public interface Presenter {
        void onCancel();

        void onJoin(String string);
    }
}
```

[gwt.xml maps Factory.class to the right type]
Benefits and tradeoffs

- Same as MVC with improved:
  - Decoupling of views from the model
    - Split teams [relieve critical path]
  - Testability [reduce UI testing]
  - A little less complex than MVC [fewer events]
Architecture/Design Review Meeting

› Don’t think of this as an oral exam
› Start with 10 minute presentation (board only)
› Followed by 40 minute discussion
› Evaluating the product, not the producer
› Be prepared!
› Goal:
  ‣ Ensure system meets proposal
  ‣ Check consistency of design with architecture
  ‣ Talk about design decisions/justification
  ‣ Discuss support for future system evolution