Desktop Patterns and Data Binding

Karsten Lentzsch
Goal

Learn how to organize presentation logic and how to bind domain data to views
Agenda

Introduction
Autonomous View
Model View Presenter
Presentation Model
Data Binding
Agenda

Introduction
Autonomous View
Model View Presenter
Presentation Model
Data Binding
Legend

Presentation (View)

Presentation Logic

Refers to

Notifies

Domain Object
Legend

- Domain/business logic
- Examples:
  - Book
  - Person
  - Address
  - Invoice
- More generally: object graph

Domain Object
Legend

- **Presentation Logic**
- **Handlers for:**
  - List selection changes
  - Check box selection
  - Drag drop end
- **UI models**
  - ListModel
  - TableModel
  - TreeSelectionModel
- **Swing Actions**
Event Handling vs. Presentation Logic

- **Toolkit handles fine-grained events:**
  - Mouse entered, exited
  - Mouse pressed
  - Radio button pressed, armed, rollover

- **Application handles coarse-grained events:**
  - Radio button selected
  - Action performed
  - List items added
  - Domain property changed
Legend

- Container:
  - JPanel, JDialog, JFrame
- Contains components:
  - JTextField, JList, JTable
- Component initialization
- Panel building code
- GUI state:
  - Check box pressed
  - Mouse over
Legend

- Role1 and Role2 “sit together” in a class
- Can access each other
- Separated layers
Legend

- A refers to B
- A holds a reference to B
- B indirectly refers to A
All Mixed Together

- Presentation (View)
- Presentation Logic
- Domain
**Pattern:** Separated Presentation

- Presentation (View)
- Presentation Logic
- Domain
Business Logic in the Presentation

- Presentation (View)
- Presentation Logic
- Business Logic

Domain
Decouple Domain from Presentation

- The domain shall not reference the presentation
- Presentation refers to domain and modifies it

Advantages:
- Reduces complexity
- Multiple presentations
Separated Presentation with **Observer**

- Presentation (View)
- Presentation Logic
- Domain

Presentation (View) Notifies Domain
Refers to Presentation Logic
Agenda

Introduction

**Autonomous View**

Model View Presenter

Presentation Model

Data Binding
Pattern: Autonomous View

- Presentation (View)
- Presentation Logic
Autonomous View

- Often one class per window or screen
- Often a subclass of JDialog, JFrame, JPanel
- Contains:
  - Fields for UI components
  - Component initialization
  - Panel building/layout
  - Model initialization
  - Presentation logic: listeners, operations
Example GUI

Composer field is enabled, if classical is selected
public class AlbumDialog extends JDialog {

    private final Album album;

    private JTextField artistField;
    ...

    public AlbumDialog(Album album) { ... }

    private void initComponents() { ... }

    private void initPresentationLogic() { ... }

    private JComponent buildContentView() { ... }

class ClassicalChangeHandler
    implements ChangeListener {

    public void stateChanged(ChangeEvent e) {
        // Check the classical state.
        boolean classical = classicalBox.isSelected();

        // Update the composer field enablement.
        composerField.setEnabled(classical);
    }
}
Autonomous View: Tips

- **Build** dialogs, frames, panels
- **Extend** JDialog, JFrame, JPanel if necessary. *Do you extend or use HashMap?*
Autonomous View

- Common and workable
- Has disadvantages:
  - Difficult to test logically
  - Difficult to overview, manage, maintain, and debug, if the view or logic is complex
- Consider to separate the logic from the view
Presentation Logic Separated

Presentation (View)  Presentation Logic

Domain
Separated Logic: Advantages I

- Allows to test the presentation logic logically
- Simplifies team synchronization
- Each part is smaller and easier to overview
- Allows to build “forbidden zones”
  - For team members
  - Before you ship a new release
    - Layout changes allowed
    - Design is done, but bug fixes in the logic are still allowed
Separated Logic: Advantages II

• Thin GUI:
  • Easier to build, understand, maintain
  • Can follow syntactical patterns
  • More team members can work with it

• Logic can ignore presentation details, e.g. component types (JTable vs. JList)
• Logic can be reused for different views
Separated Logic: Disadvantages

- Extra machinery to support the separation
- Extra effort to read and manage multiple sources
Separating Logic from the View

- Can simplify or add complexity
- Separation costs vary with the pattern used
- **Opinion**: typically you benefit from the separation

My personal guideline for team projects:
- Use Autonomous View for message dialogs
- Otherwise separate the logic from the view
Agenda

Introduction
Autonomous View
Model View Presenter
Presentation Model
Data Binding
Pattern: Model View Presenter (MVP)
Model View Presenter

View
- Holds UI components
- Holds GUI state
- Inits components
- Builds panel

Presenter
- Reads domain data
- Sets GUI state
- Presentation Logic
- Changes the domain

Model
- Holds domain data
- Provides business logic
Album Example: Autonomous View

AlbumDialog

- JTextField
- JTextField
- JCheckBox
- JTextField

ChangeHandler

Album
Album Example: Model View Presenter

AlbumView
- JTextField
- JTextField
- JCheckBox
- JTextField

AlbumPresenter
- ChangeHandler

Album
From Autonomous View ...

```java
public class AlbumDialog extends JDialog {
    private JTextField artistField;
    public AlbumDialog(Album album) { ... }
    private void initComponents() { ... }
    private JComponent buildContent() { ... }

    private final Album album;
    private void initPresentationLogic() { ... }
    private void readGUIStateFromDomain() { ... }
    private void writeGUIStateToDomain() { ... }
    class ClassicalChangeHandler implements ...
    class OKActionHandler implements ...
}
```
... to Model View Presenter

class AlbumView extends JDialog {
    JTextField artistField;
    public AlbumView() { ... }
    private void initComponents() { ... }
    private JComponent buildContent() { ... }
}

public class AlbumPresenter {
    private final AlbumView view;
    private Album album;
    private void initPresentationLogic() { ... }
    private void readGUIStateFromDomain() { ... }
    private void writeGUIStateToDomain() { ... }
    class ClassicalChangeHandler implements ...
    class OKActionHandler implements ...
}
... to Model View Presenter

class AlbumView extends JDialog {
    JTextField artistField;
    public AlbumView() { ... }
    private void initComponents() { ... }
    private JComponent buildContentView() { ... }
}

public class AlbumPresenter {
    private final AlbumView view;
    private Album album;
    private void initPresentationLogic() { ... }
    private void readGUIStateFromDomain() { ... }
    private void writeGUIStateToDomain() { ... }
    class ClassicalChangeHandler implements ...
    class OKActionHandler implements ...
}
Presenter: Example Logic

class ClassicalChangeListener
    implements ChangeListener {

    public void stateChanged(ChangeEvent e) {
        // Check the view's classical state.
        boolean classical =
            view.classicalBox.isSelected();

        // Update the composer field enablement.
        view.composerField.setEnablement(classical);
    }
}
Agenda

Introduction
Autonomous View
Model View Presenter
Presentation Model
Data Binding
Pattern: Presentation Model

View

Presentation Model

Domain
Presentation Model

View

- Holds UI components
- Holds all GUI state
- Inits components
- Builds panel
- Listens to PM changes

Presentation Model

- Reads domain data
- Holds relevant state
- Presentation Logic
- Fires state changes
- Changes the domain

Domain
Reminder: Swing Actions

**JButton**
- Holds all GUI state
- Listens to Action changes

**Action**
- Holds *relevant* state
- Fires state changes
public class AlbumDialog extends JDialog {
    private JTextField artistField;
    public AlbumDialog(Album album) { ... }
    private void initComponents() { ... }
    private JComponent buildContent() { ... }

    private final Album album;
    private void initPresentationLogic() { ... }
    private void readGUIStateFromDomain() { ... }
    private void writeGUIStateToDomain() { ... }
    class ClassicalChangeHandler implements ...
    class OKActionHandler implements ...
}
... to Presentation Model

class AlbumView extends JDialog {
    private final AlbumPresentationModel model;
    private JTextField artistField;
    public AlbumView(AlbumPM model) {
        ... }
    private void initComponents() {
        ... }
    private JComponent buildContent() {
        ... }
}

public class AlbumPresentationModel {
    private Album album;
    private void initPresentationLogic() {
        ... }
    private void readPMStateFromDomain() {
        ... }
    private void writePMStateToDomain() {
        ... }
    class ClassicalChangeHandler implements ...
    class OKActionHandler implements ...
}
AlbumPresentationModel

AlbumView

- JTextField
- JTextField
- JCheckBox
- JTextField

AlbumPM

- Text Model
- Text Model
- Selection Model
- Text Model

Album
AlbumPresentationModel: Logic

AlbumView
- JTextField
- JTextField
- JCheckBox
- JTextField
- ChangeHandler

AlbumPM
- Text Model
- Text Model
- Selection Model
- Text Model
- ChangeHandler
- Enablement Model

Album
AlbumPresentationModel: Logic

**AlbumView**
- JTextField
- JTextField
- JCheckBox
- JTextField

**ChangeHandler**

**AlbumPM**
- Text Model
- Text Model
- Selection Model
- Text Model
- ChangeHandler
- Enablement Model

**Album**
AlbumPresentationModel: Logic

AlbumView
- JTextField
- JTextField
- JCheckBox
- JTextField

ChangeHandler

AlbumPM
- Text Model
- Text Model
- Selection Model
- Text Model
- ChangeHandler
- Enablement Model

Updates

Album
AlbumPresentationModel: Logic

AlbumView
  JTextField
  JTextField
  JCheckBox
  JTextField

ChangeHandler

Notifies

AlbumPM
  Text Model
  Text Model
  Selection Model
  Text Model
  ChangeHandler

Enablement Model

Album
AlbumPresentationModel: Logic

AlbumView
- JTextField
- JTextField
- JCheckBox
- JTextField

Updates enablement

ChangeHandler

AlbumPM
- Text Model
- Text Model
- Selection Model
- Text Model
- ChangeHandler

Enablement Model

Album
No Worries: Actions Again

- Swing uses a similar machinery for Actions
- Actions fire PropertyChangeEvent
- JButton listens to the Action and updates its state
- Swing synchronizes Action state and GUI state
- All **you** need to write is:
  
  ```java
  new JButton(anAction)
  ```
Action with Multiple Views

JButton

Action
• Text
• Icon
• Enablement
• Mnemonic

JButton

JMenuItem
Presentation Model: Multiple Views I

Panel with List and Button
- JList
- JButton

PopupMenu
- JMenuItem

Presentation Model
-ListModel
- Action

Domain
Presentation Model: Multiple Views II

Presentation Model

Display List
- JList

Table with Button
- JTable
- JButton
- TableModelAdapter

ListModel

Action

Domain
MVP vs. Presentation Model: GUI State

- MVP
  - View holds the GUI state
  - Presenter holds no state
  - Avoids having to synchronize copied GUI state

- Presentation Model
  - View holds all GUI state
  - PM holds the relevant GUI state
  - Must synchronize PM state and View state
MVP vs. Presentation Model: Testing

- MVP
  - Allows to test the Presenter with a View stub
  - Allows to preview the View without the Presenter

- Presentation Model
  - Allows to test the Presentation Model without the View
  - Allows to preview the View with a PM stub
MVP vs. Presentation Model: Transformation Differences

- Some Autonomous Views use low-level GUI state
- Presenter can keep “dirty” low-level ops
  - Split to MVP is easier to do
  - Split to MVP may costs less
- Split to PM may require extra work
  - Find and add GUI state abstractions
  - Add handlers to the view
- You may benefit from the extra cleaning
MVP vs. Presentation Model: General

- Developers are used to operate on view state
- Presenter depends on GUI component types
- MVP addresses problems many faced with PM
Agenda

Introduction
Autonomous View
Model View Presenter
Presentation Model
Data Binding
Data Binding

- Synchronizes two data sources
- One-way or two-way
- Typically supports type conversion
- May provide a validation
Binding Examples

![Diagram showing binding examples]

View
- JButton
- JTable

Presentation Model
- Action
- TableModel
Binding Examples

View

JCheckBox
selected=true

JTextField
enabled=true

Album
classical=true
Binding Examples

GUI Form

- JTextField
- JCheckBox
- JFormattedTextField

Database
Useful Swing Bindings

View

- JList
- JTable
- JTree
- JButton

Presentation Model

- ListModel
- TableModel
- TreeModel
- Action
Swing Binding to Low-Level Models

![Diagram of Swing Binding to Low-Level Models]

- **View**: JTextField, JCheckBox, JFormattedTextField
- **Presentation Model**: Document, ToggleButtonModel, Document
Wanted: Higher-Level Binding

View

- JTextField
- JCheckBox
- JFormattedTextField

Presentation Model

- Text Model
- Boolean Model
- Date Model
Wanted: Full Binding Path

View

- JTextField
- JCheckBox
- JFormattedTextField

Presentation Model

- Text Model
- Boolean Model
- Date Model

Album

- artist="John"
- classical=true
- released=05/16/06
JGoodies Binding

• Uses Swing bindings:
  • JList, JTable, JComboBox, JTree, JButton
• Fills the gap where Swing uses low-level models:
  • JTextField, JCheckBox, ...
• Converts Bean properties to a uniform model (ValueModel)

• Makes the hard stuff possible
• Makes simple things a bit easier
private void initComponents() {
    artistField = Factory.createTextField(
        presentationModel.getModel("artist"));

    classicalBox = Factory.createCheckBox(
        presentationModel.getModel("classical"));

    songList = Factory.createList(
        presentationModel.getSongsAndSelection());

    okButton = new JButton(
        presentationModel.getOKAction());
}
AlbumView: EnablementHandler

private void initPresentationLogic() {

    // Synchronize field enablement
    // with the PresentationModel state.
    PropertyConnector.connect(
        presentationModel,
        "composerEnabled",
        composerField,
        "enabled");

}
JSR 295: Beans Binding

- Synchronizes a data source with a target (often two bound bean properties)
- Shall support type conversion and validation
- Has a BindingContext as a container for multiple bindings
Copying ...

- Easy to understand
- Works in almost all situations
- Easy to debug; all data operations are explicit
- Difficult to synchronize views
- Needs discipline in a team
- Coarse-grained updates
- Leads to a lot of boilerplate code
... vs. Automatic Binding

- Fine-grained updates
- Simplifies synchronization
- Harder to understand and debug
- Extra work for method renaming and obfuscators
Costs for Automatic Binding

- Increases learning costs
- Decreases production costs a little
- Can significantly reduce the change costs
Summary

- Starting point: **Separated Presentation**
- Common and workable: **Autonomous View**
- **MVP** works with view GUI state
- **PM** copies state and requires synchronization
- Swing has some **Presentation Model** support
Advice

- Use *Separated Presentation* whenever possible
- Split up *Autonomous Views* if appropriate
- Read Fowler's “Organizing Presentation Logic”

- Use an automatic binding only if
  - it's reliable and flexible
  - at least one expert in the team masters it
For More Information

Web Resources

• Fowler's Further P of EAA – martin Fowler.com/eaaDev
  
• SwingLabs data binding – databinding.dev.java.net
• Eclipse 3.2 data binding – www.eclipse.org
• Oracle ADF – otn.oracle.com, search 'JClient'
• JGoodies Binding – binding.dev.java.net
  Binding tutorial contains Presentation Model examples
• JSR 295 Beans Binding – jcp.org/en/jsr/detail?id=295
For More Information

Book
  • *Scott Delap: Desktop Java Live*

Presentations - [www.JGoodies.com/articles](http://www.JGoodies.com/articles)
  • Desktop Patterns & Data Binding
  • Swing Data Binding