Advanced Swing GUI Programming

Kyle Brown
Executive Java Consultant
IBM WebSphere Services
RTP, N.C
brownkyl@us.ibm.com
ICM Architecture and Swing

GUI Builders and Swing

In-depth Video Store Example
  ▲ Table models
  ▲ List Models
Layered Architecture

- Compared to O O Design, little has been written about O O Architectures
  - An Architecture is a general layout of a program.
  - Shows how all the pieces fit together

- A Layered architecture is the most common
  - Provides built-in enhanceability and easy reuse.
  - Provides a structure for project planning and estimation

- An exception would be the remarkable work of Shaw and Garlen. Also included (IMHO) would be the patter work by Buschmann, et. al.
- Layered architectures are common in telecommunications (the OSI model), operating systems (e.g. Mach and Kernel architectures) and control systems.
What is a layer?

- A Layer is a set of interacting objects that perform a common set of tasks
  - presents a unified interface to the outside world

- Objects in a layer may collaborate with objects that are in layers below them
  - Objects should not collaborate with objects in higher layers.
  - They may collaborate with other objects in the same layer
ICM is a simple layered architecture for business systems
Separates the Interface from the Control from the Model

ICM Architecture
Layer Definitions

- **Interface**
  - Where windows (Frames) live. Contains GUI Components from an application builder palette. Often generated code.

- **Control Layer**
  - Serves to adapt the protocol of the GUI layer to that of the domain layer.
  - Manages the interaction of the pieces of an interface
Layer Definitions (2)

- Domain model Layer
  - The meat of the application
  - Most of the objects found in OOA&D.

- Infrastructure Layer
  - supports the domain model
  - databases, comms, etc.
Why do you need a Control layer?

- Many tools like Visual Basic do not support a separate Application layer
  - they mix GUI code with domain code indiscriminately

- So why do you need it?
  - It separates the semantics of the application from the details (syntax) of the particular GUI
  - It allows you to implement the same application with different interfaces
    - Applets, Applications, Servlets
Applications should be built like a sandwich
Start with the meat, then work out to the bun.
Application Building Steps

- **Step 1:** Build an initial domain layer
  - code the domain objects first
  - test them in-place with a test skeleton

- **Step 2:** Build an initial GUI
  - draw the appropriate windows
  - or construct them from scratch (painful!)

- In step 1 you can make yourself certain your business logic is correct and that your abstractions are the right ones
- In an iterative process you stop here and assess risks for the next stage -- often involves reworking your design!
- In step 2 you will usually find that you missed some model abstractions -- iterate!
Application Building Steps (2)

- Step 3: Build a Control layer
  - utilize the Adapter and Mediator patterns
  - utilize the rules for Controllers
  - these should be separate objects from the Applets or Frames except in VERY trivial cases

- Step 4: Build the infrastructure layer
  - start small
  - microlayer the infrastructure

- Step 5: Go back to Step 1 and Iterate!
Rule 1: Keep your app models small and thin
- few instance variables (< 3 if possible)
- most state should be kept in the domain model
- as few methods as possible

A layered approach encourages this. In general, methods should be very short; if a method is getting long it probably belongs in the domain model.
Rule 2: Keep your app models dumb!

- Mediate and Adapt
  - do not take over the Model’s responsibilities!

- If it doesn’t mediate or adapt, move it out!
  - do not take over the Model’s responsibilities!
GUI Builders

While it is possible to build Swing interfaces by hand I don't recommend it

N o human should have to understand GridBagLayout

Many GUI builder tools (VisualAge for Java, Visual Cafe, Forte for Java) can handle building Swing GUI's

you can mix generated code (for simple things) with hand-written code for complex things

An easy trap to fall into is to let your GUI builder try to do everything! Unfortunately, many GUI builders subtly encourage this.

Remember you want to LAYER your application. Just draw the window in the GUI builder and then drop down to code for the guts of the application.
My advice -- ignore all of the visual programming parts. Don't use the data access beans or enterprise beans in visual programming.

Just use the Visual Composition Editor for page layout, and then have code for event handlers call down into the Controller layer.

This is usually simpler and more easily ported.
Now we will examine ICM in the context of the Video Store Example

Basic design issues
- Controller Design issues
- Using a GUI builder with Swing
- Advanced Swing Programming issues
Layers and Package names

- Package names should be based on the layer they contain
  - Also include standard organization, project information in the name
  - See [Woolf] for more details

<table>
<thead>
<tr>
<th>edu.ncsu.examples.video.ui</th>
<th>edu.ncsu.examples.video.tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>edu.ncsu.examples.video.model</td>
<td></td>
</tr>
<tr>
<td>edu.ncsu.examples.video.factories</td>
<td></td>
</tr>
</tbody>
</table>
Video Store Design Overview

- VCCustomerFactory
- VCVideoTapeFactory
- VCMovieFactory
- VCCategoryFactory

- VCCustomer
  - VCAddress
  - VCCustomerTransaction
    - VCTapeRental
    - VCMovie
      - VCCategory
    - VCVideoTape
In our case, each UI knows its associated controller held in a variable named "controller" added to the Panel.

The Panels receive event notifications, pass control to the controllers keeps the Interface layer from implementing model or control layer behavior Panels communicate with controllers in terms of simple Java types.
public class VideoReturnPanel extends JPanel {
    public VideoReturnController controller = new VideoReturnController();
    ...
    public void returnButtonPressed() {
        // This message is invoked by an ActionEvent Handler (not shown)
        String tapeId = getTapeNumberTextField().getText();
        try {
            getController().returnVideo(tapeId);
            getMessageLineLabel().setText("Tape " + tapeId + " returned.");
        } catch (edu.ncsu.examples.video.model.DuplicateTapeReturnException e) {
            getMessageLineLabel().setText("Cannot return duplicate tape; Check number.");
        }
    }
    ...
}
/* The tape id obtained from the tape id TextField is passed in as a String */
public void returnVideo(String tapeId) throws DuplicateTapeReturnException {
    int id = Integer.parseInt(tapeId);
    VCVideoTape tape = VCVideoTapeFactory.getInstance().getTapeForId(id);
    tape.returnTape();
}
MovieSearchPanel

JLabel

MovieTitleEntryField (JTextField)

Search Button (JButton)

JScrollPane

TapeListingTable (JTable)

TapeNumberColumn (TableColumn)

LocationColumn (TableColumn)

DueDateColumn (TableColumn)

RemoveButton (JButton)
private javax.swing.JTable getTapeListingTable() {
    if (ivjTapeListingTable == null) {
        ivjTapeListingTable = new javax.swing.JTable();
        ivjTapeListingTable.setName("TapeListingTable");
        getJScrollPane2().setColumnHeaderView(ivjTapeListingTable.getTableHeader());
        getJScrollPane2().getViewport().setBackingStoreEnabled(true);
        ivjTapeListingTable.setAutoResizeMode(0);
        ivjTapeListingTable.setPreferredSize(new java.awt.Dimension(200, 200));
        ivjTapeListingTable.setBounds(0, 0, 200, 200);
        ivjTapeListingTable.setAutoCreateColumnsFromModel(false);
        ivjTapeListingTable.setPreferredSize(new java.awt.Dimension(200, 200));
        ivjTapeListingTable.setBounds(0, 0, 200, 200);
        ivjTapeListingTable.setAutoCreateColumnsFromModel(true);
        ivjTapeListingTable.addColumn(getTapeNumberColumn());
        ivjTapeListingTable.addColumn(getLocationColumn());
        ivjTapeListingTable.addColumn(getDueDateColumn());
        // user code begin {1}
        ivjTapeListingTable.setModel(getController().getTableModel());
        // user code end
    } return ivjTapeListingTable;
}
private javax.swing.table.TableColumn getLocationColumn() {
    if (ivjLocationColumn == null) {
        ivjLocationColumn = new javax.swing.table.TableColumn();
        ivjLocationColumn.setModelIndex(1);
        ivjLocationColumn.setWidth(104);
        ivjLocationColumn.setHeaderValue("Location");
        // user code begin {1}
        // user code end
    }
    return ivjLocationColumn;
}
public class TapeTableModel extends AbstractTableModel {
    public VCMovie movie = new VCMovie();

    /* Would have getters/setters for movie */

    public int getRowCount() {
        if (getMovie() != null)
            return movie.getTapes().size();
        else
            return 0;
    }
}
public Object getValueAt(int row, int col) {
    VCVideoTape tape = (VCVideoTape) getMovie().getTapes().elementAt(row);
    switch (col) {
    case 0: // tape number
        return Integer.toString(tape.getNumber());
    case 1: // rented or on shelf
        if (tape.isRented()) return "Rented";
        else return "On Shelf";
    case 2: // due back date
        if (tape.isRented()) {
            java.util.Date rDate = tape.getCurrentRental().getDueDate();
            return java.text.DateFormat.getInstance().format(rDate);
        }
        else return "N/A";
    default: // should never happen
        return null;
    }
}
public void searchButtonPressed() {
    // called from an ActionEvent handler (not shown)
    String entry = getMovieTitleEntryField().getText();
    controller.findMovie(entry);
    refresh();
}

public void refresh() {
    getTapeListingTable().repaint();
}
public void findMovie(String entry) {
    try {
        VCMovie movie = VCMovieFactory.getInstance().findMovieForTitle(entry);
        getTableModel().setMovie(movie);
    } catch (NoSuchElementException e) {
        System.out.println("no movie found");
    }
}
JOptionPane is the Swing "MessageBox" class
- It allows you to construct simple dialogs with OK, YES, NO, CANCEL options
- Also allows simple selection from a drop-down

There are multiple show(Input,Confirm)Dialog methods to display these
- take as an argument the combination of button options
/**
 * Called from an ActionEvent handler (not shown)
 */

public void removeButtonPressed() {
    JRootPane root = getRootPane();
    JOptionPane option = new JOptionPane();
    int selection =
            option.showConfirmDialog(
                root,
                "This operation cannot be undone. Proceed?",
                "Remove video tape",
                option.YES_NO_OPTION);
    if (selection == option.YES_OPTION) {
        int index = getTapeListingTable().getSelectedRow();
        getController().removeTape(index);
    }
}
MovieSearchController Code

```java
public void removeTape(int index) {
    getTableModel().getMovie().removeMovie(index);
    getTableModel().removeElement();
}
```

TapeTableModel Code

```java
public void removeElement() {
    fireTableRowsDeleted(-1, -1);
}
```
private javax.swing.JList getLineItemList() {
    if (ivjLineItemList == null) {
        ivjLineItemList = new javax.swing.JList();
        ivjLineItemList.setName("LineItemList");
        ivjLineItemList.setBounds(0, 0, 160, 120);
        // user code begin {1}
        ivjLineItemList.setModel(getController().getListModel());
        // user code end
    }
    return ivjLineItemList;
}

public void findCustomerButtonPressed() {
    String customerNumber = getCustomerNameTextField().getText();
    getController().findCustomer(customerNumber);
    refresh();
}
public void refresh() {
    // force the list to refresh.
    getLineItemList().repaint();
    // force the sales tax to refresh
    String formattedTax = getController().getSalesTax();
    getSalesTaxLabel().setText(formattedTax);
    // force the total to refresh
    String formattedTotal = getController().getTotalDue();
    getTotalLabel().setText(formattedTotal);
    // force the customer name to refresh
    String customerName = getController().getCustomerName();
    getCustomerNameLabel().setText(customerName);
}
public void findCustomer(String customerNumber) {
    int cNum = Integer.parseInt(customerNumber);
    VCCustomer customer =
        VCCustomerFactory.getInstance().getCustomerForId(cNum);
    setCustomer(customer);

    VCCustomerTransaction tran = new VCCustomerTransaction();
    tran.setCustomer(customer);
    tran.getOverdueTapes(); // add overdue tapes immediately
    setTransaction(tran);
    getModel().setTransaction(tran);
}
CheckoutController code

/* This is invoked from the Panel similarly to the previous examples */

code

```java
public void addTape(String tapeNumber) {
    int tapeNum = Integer.parseInt(tapeNumber);
    getTransaction().addTapeForId(tapeNum);
    getListModel().addElement(null);
}
```

CheckoutListModel code

```java
public void addElement(Object element) {
    fireContentsChanged(this, -1, -1);
}
```

© Copyright IBM Corporation 1999
public int getSize() {
    if (getTransaction() != null) {
        return getTransaction().getRentals().size();
    } else {
        return 0;
    }
}
public Object getElementAt(int index) {
    VCTapeRental rental = (VCTapeRental) getTransaction().getRentals().elementAt(index);
    StringBuffer buff = new StringBuffer(50);
    buff.append(NumberFormat.getCurrencyInstance().format(rental.getFeeOrFine()));
    if (rental.isOverdue()) {
        buff.append(" OVERDUE FINE -- ");
        buff.append(rental.getTape().getMovie());
        buff.append(" ");
        buff.append(rental.getOverdueDays());
        buff.append("DAYS LATE");
    } else {
        buff.append(" RENTAL ");
        buff.append(rental.getTape().getMovie().getTitle());
        buff.append(" DUE BACK ");
        buff.append(DateFormat.getInstance().format(rental.getDueDate()));
    }
    return buff.toString();
}
Sometimes a bidirectional UI/controller link is necessary
- Want to minimize coupling between layers
- Should try to make it unidirectional if you can
Often Controllers will have subcontrollers

- Parallels a Frame/Panel relationship
- Allows the top Controller to delegate some work
We've seen:
- How Layering is applied in the ICM architecture
- An example of ICM with Swing in the Video Store
References

- Erich Gamma, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995
- Kyle Brown, "Distributed ICM", Distributed Object Computing, October, 1997 available at http://members.aol.com/kgb1001001
Bobby Woolf, "Partitioning Smalltalk Code into Envy/Developer Components", Pattern Languages of Program Design 2, Addison-Wesley, 1996