UI software architectures & Modeling interaction

(part of this content is based on previous classes from A. Bezerianos, S. Huot, M. Beaudouin-Lafon, N.Roussel, O.Chapuis)

Software architecture - MVC

Assignment 1

Design and implement an interactive tool for creating the layout of comic strips


structure of an interactive system

What we see
- output

What we act with
- input

What happens
- treatment
- computation
- communication
- data (storage and access)

visible part « front end »

invisible part « back end »
example 1
- data model (albums, artists, categories, etc.)
- communication with iTunes server
- manage queries
- manage sales
- security

example 2
- geometric models
- calculations (transformations, rendering, etc.)
- store and access designs

example 3
- tabular structure
- storage and data access

link between the two parts
... programming using an organization model
organize, structure an interactive application by separating:
- Data and their treatment: the Model
- Data representation: the View
- Application behavior to input: the Controller
Model «Model–View–Controller» (MVC)

MVC is:
- A design pattern (standardized design solution independent of programming language)
- A software architecture (a way to structure an application or a set of software packages)

Introduced in 1979 by Trygve Reenskaug

Strongly linked to OO programming (Smalltalk)

MVC: ideal interactions between components

MVC: interactions between components

MVC: interactions between components
**MVC: interactions between components**

- **Model**
  - Application functionality
  - Data access and management

- **View**
  - Presentation of data and functionality to the user
  - Notification of input

- **Controller**
  - Manage user input
  - Update application behavior
  - Notification of state change

- **User input**
MVC: the model

The model:
- Represents data
- Gives access to data
- Gives access to data management functionality
- Exposes the application functionality

Functional layer of the application

MVC: the view

The view:
- Shows the (or one) representation of the data in the model
- Ensures consistency between data representation and their state in the model (application)

Output of the application

MVC: the controller

The controller:
- Represents the application behavior w.r.t. user actions
- Translates user actions to actions on the model
- Calls the appropriate view w.r.t. the user actions and the model updates

Effect and treatment of input

advantages of MVC

Clean application structure
Adapted to concepts of O-O programming
**Independence** of data – representation – behavior
**Modular** and **reusable**
disadvantages of MVC

- Implementation complex for large applications
- Too many calls between components
  - "Spaghetti" code
- Controller and View are often tightly linked to Model (and often to each other)

need to adapt implementation

MVC and Java Swing Widgets

- Model-View-Controller separation not strict
- Model categories:
  - Visual status of GUI controls, e.g., pressed or armed button
  - Application-data model, e.g., text in a text area
- Swing uses a model by default for each widget
- View & Controller (often part of the same UI object)
  - Look & Feel + Listener
  - Examples: JButton, JLabel, JPanel, etc.

example

```java
Object[][] data = {
    {"Kathy", "Smith", "Snowboarding", new Integer(5), new Boolean(false)},
    {"John", "Doe", "Rowing", new Integer(3), new Boolean(true)},
    {"Sue", "Black", "Knitting", new Integer(2), new Boolean(false)},
    {"Jane", "White", "Speed reading", new Integer(20), new Boolean(true)},
    {"Joe", "Brown", "Pool", new Integer(10), new Boolean(false)}
};
```

example

```java
javax.swing.JTable
javax.swing.table.TableModel
```

The data
The model

class MyTableModel extends AbstractTableModel {
    private String[][] columnNames = ...;
    private Object[][] data = ...;
    public int getColumnCount() {
        return columnNames.length;
    }
    public int getRowCount() {
        return data.length;
    }
    public String getColumnName(int col) {
        return columnNames[col];
    }
    public Object getValueAt(int row, int col) {
        return data[row][col];
    }
}

example

The view

TableModel dataModel = new MyTableModel();
JTable table = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(table);

example

The controller

public class MySelectionListener implements ListSelectionListener {
    private JTable table;
    public MySelectionListener(JTable table) {
        this.table = table;
        table.setCellSelectionEnabled(true);
        ListSelectionModel cellSelectionModel = table.getSelectionModel();
        ListSelectionModel tableSelectionModel = table.getSelectionModel();
        cellSelectionModel.addListSelectionListener(this);
    }
    public void valueChanged() {
        ...
    }
}
WIMP interfaces

WIMP: Window, Icons, Menus and Pointing

Presentation
- Windows, icons and other graphical objects

Interaction
- Menus, dialog boxes, text input fields, etc

Input
- pointing, selection, ink/path

Perception-action loop
- feedback

direct manipulation

Ben Shneiderman (1983)

1. Persistent representation of objects of interest
2. Use of physical actions instead of complex syntax
3. Operations are quick, incremental, reversible, and their effect on objects is immediately visible (feedback)
4. Incremental learning, to permit use of the interface with little prior knowledge

direct manipulation: examples

- text editors (e.g., Word, OpenOffice)
- bitmap/vector graphics (e.g., Photoshop, Illustrator).
- Counter-example: Latex...

Icon interaction:
- Generic interface
- Use of metaphors
- drag-and-drop

direct manipulation?
direct manipulation problems

Identifying objects of interest
- example: styles in Word

Immediate feedback difficult when there is a delay between action and result

Direct or indirect manipulation?
- menus, dialog boxes, scroll-bars, etc.

describing interactions: state machines

Finite Automata
State = interaction state
Transition = input events

State Machine
- boolean expressions of events associated to transitions (guard)
- actions associated to transitions (not always present)

Example:

example: dragging windows

Example:
state machines & MVC

Common approach: use of global variables within a controller

```java
public enum State {S1, S2, S3, S4}
private State state = State.S1;
```

or (use of multiple variables)

```java
private boolean buttonPressed = false, mouseMoved = false;
```

In the following lecture, we'll introduce SwingStates, a Java library for modeling interaction through states, state transitions, and state machines.

common problems

Getting trapped to states with no transitions (deadlocks)

Maintaining the code to capture new or unforeseen states is usually hard

An interaction can involve several UI components. Not always clear how to divide interaction between multiple controllers and state machines.

drag & drop

Which UI objects are involved?
Which controller handles this interaction?
interaction modes

Mode: distinct state of the UI where the same user input has a different interpretation
- text vs. drawing mode in an editing tool
- typing capital or small characters

Mode switching
- e.g., Caps lock key, specialized button

Quasimode: mode being active through some constant action from the user
- e.g., use of modifier keys such as Shift, Alt, Control while typing or pointing

interaction modes: problems

« modes are a significant source of errors, confusion, unnecessary restrictions, and complexity in interfaces »

Jef Ruskin

Ruskin advocated for modeless interfaces. He also recommended the use of quasimodes instead of explicit modes.

Other points of view (Jacob Nielsen)
- « users cannot cope with everything at once »
- « ...need the interface to narrow their attention »
- « Real life is highly moded »

making modes visible

eliminating modes

Special mode for changing time  No modes, direct editing

What are the trade-offs in these designs?